Data security aspects of a debit card reader system

Jacobus Theron Botha

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg in partial fulfillment of the requirements for the degree of Master of Science in Engineering.

Johannesburg 1990

I declare that this project report is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

JF Botha

20th day of September 1990

Abstract

A *debit-card* is a form of payment. The card is pre-charged with a value, monetary or otherwise, before distribution to the user, and is therefore a *pre-payment card* . It is then 'used up', its value being decremented until it is valueless. At this point, it is either discarded or re-charged by the administration authority. It is distinct from a *credit card*, which provides a post-payment facility.

The aim of this project is to investigate the security aspects of a debit-card reader system for use as an unattended fee-collecting subsystem in such applications as public telephones, parking meters and vending machines.

Card technology and attributes of debit card systems are discussed, an overview of cryptology is given, and an implementation of a magnetic card system is described.

Acknowledgments

This project was carried out under the auspices of Telkor Research and Development in Sandton, South Africa.

The card reader electronics circuitry was designed and developed by Dawood Kadwa, and the card reader software was designed and developed by Joanne Carey. The card reader mechanism is under development by Jack Cook and Mark Richardson.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Applications

It is envisaged that a debit card reader may be used wherever a user needs to make payment to an unattended fee-collecting device. Applications which have been mooted for the card reader under development are the following:

- Phone cards. The user will purchase a card of a specific value at a post office or elsewhere, and be able to use public payphones.
- Photocopier enablers, where the card will be charged with a number of copies. The card may be required to be valid only at one machine, or at a range of machines.
- Electricity enablers. Here the user will be required to purchase a card having a number of kW-Hr units encoded on it, and the mains supply to his domicile will be enabled only by the card-operated device.
- Parking meters. Users will insert their parking card into the meter, which will display the parking time thus purchased.
- Vending machines for soft drinks, cigarettes and food items.
- Rail tickets which can gain access to a turnstile.

The intention is to develop the card reader as a unit which will operate in conjunction with a host system and which will perform the functions of

- reading data from a card,
- verifying the validity of the card,
- informing the host of the value and type of card, and
- writing a new value back to the card upon the host's request.

1.2 Existing Card Technology

1.2.1 Card Formats

Cards for payment have evolved through several formats. An outline, in order of advancement of technology, is as follows:

Mechanical or Thermal This is the oldest type of card, and includes the old-fashioned bus and train ticket which is punched by an inspector to indicate that it has been used.

Magnetic stripe cards Among these cards are the *low-coercivity* types, which are prone to disturbance by magnetic fields, because a relatively weak magnetic field is needed to orient their magnetic domains. Such cards employ magnetic tape which can be read and written by commonly available magnetic heads such as those used in audio cassette players.

For financial transaction cards and identification cards ISO standard 7811 – 7813 [40] and ANSI Standard X4.16 [30] govern the physical dimensions (85,72 x 54,03 x 0.76 mm), location of the magnetic stripe (11.89 mm wide, and 5.54 mm from the edge), deformation properties, resistance to chemicals, and many other characteristics. Three magnetic tracks are provided for: tracks 1 and 2 are read-only tracks, while track 3 may contain read-write data.

High-coercivity cards, requiring less easily available heads capable of generating stronger magnetic fields, also exist.

An example of such a high-coercivity system [22] is the *GEC Traffic Automation* parking meter card where *three-level encoding* is used. This means that the un-magnetized 'neutral' state is used as a third level, distinct

from the other two saturated 'north' and 'south' orientations. Regions programmed in this way are considered to be 'read-only' fields because the head cannot be used to de-magnetize an already saturated 'north' or 'south' magnetic area to a neutral state, and so cannot be used to alter the data in the three-level area fraudulently. Thus these regions can contain authenticating information.

The *Magnetic Watermark* system by *Thorn EMI Electronics* in the UK expands upon this idea by writing a permanent analog magnetic 'signature' to the magnetic stripe upon manufacture. This signature is verified by the card reader to authenticate the card presented and is used in the *Autelcard* card reader by the Swiss company *Autelca* [41].

Optical memory cards These are also called *lasercards* or *hologram cards*. In a card reader manufactured by the Swiss company *Sodeco* a holographic image is stored in a stripe on the card. The diffraction from a monochromatic light source directed at the hologram is analyzed by image-processing techniques, and the hologram is destroyed thermally to decrease the value of the card. The unused area is visibly different from the erased area, thus giving an indication of the value remaining on the card.

Chip-memory cards These are cards which contain integrated-circuit memory, i.e. RAM, EPROM, EEPROM [16, p 48] or one-shot PROM. The EPROM types usually use successive portions of the memory to write new data until all the available memory has been used up, at which time the card must be replaced.

An example is the telephone card by the French company *Flonic* [16, p 46]. Memory cards are also available in up to 2 Megabyte versions from *Epson*, *Fujisoku*, *Dallas Semiconductor*, *Du Pont Connector Systems* and *Mitsubishi Electronics America Inc.* [18, p 71].

Fused link card Such cards are manufactured by the Israeli company *Tadiran*. These are functionally similar to the chip-memory card, in that electronic fuses are blown as the card is used up.

Contacted smart card These cards contain a microprocessor and memory, as well as electrical contacts for exchanging information with the reader.

The advantages offered by smart cards include

- *on the card*-authentication. Private information, not available to unauthorized parties, can be carried on the card. A Personal Identification Code (PIN) can be a part of this restricted data and can assist in bearer identification.
- Encryption and decryption algorithms can be contained in the card memory.
- Data capacities of up to 64 k are available
- The computational abilities of the card afforded by the microprocessor offer flexibility and are available to the system administrators to provide a wide range of services such as statistics, currency conversion and calculations.

Smart Card International Inc. [18, p 72] offer a card with 8-bit microprocessor, 3 k ROM and 2 k EEPROM, together with facilities which make it compatible with existing magnetic credit card application, i.e. magnetic stripe and embossing area.

Other manufacturers of smart cards include *Honeywell Bull*, *Flonic-Schlumberger*, *Philips SA* and *Catalyst Semiconductor, Inc.*

A South African example, the utilization of a smart card system to control financial payment at a golf club, implemented by *Allied Vector Systems* using the French *Honeywell-Bull* smart card, is described in [39].

Contactless smart card This type of card sports a microprocessor and memory, but does not require electrical contacts for communication with the card reader. Instead, an RF circuit provides the communication link.

GEC Card Technology in the UK have developed such a card [23]. The card reader is equipped with an antenna which irradiates the card. The on-card RF circuitry uses this energy to power the internal semiconductor devices, and provides a serial communication channel at 9600 baud by means of RF impedance variation. At present the card does not comply with the ISO thickness specification of 0.76 mm, but a version that does is under development. *GEC* offer a software development system for this card, which includes DES routines (see chapter 2) for encryption, decryption and key

change. They report that these routines occupy about 1 k of code, and that it takes about 0.4 second to encrypt/decrypt an 8-byte block [28].

Another example is the *Tadicard* from *Tadiran* in Israel [26]. This card does not comply with the ISO thickness specifications, being about 5 mm thick.

Supersmart card In addition to the facilities of the smart card, this type features a display and keypad, and is currently the most high-tech card type. Such a card has been developed by *Toshiba* and was due to start field trial by *Visa* at the end of 1989. This card features

- an 8-bit microprocessor
- 16 k of ROM
- 8 k of RAM
- an LCD display
- a 20-key keypad
- a set of electrical contacts for serial communication with the reader
- a magnetic stripe for credit or cash card operation
- a milled credit card number for credit card voucher printing. The card number is milled rather than embossed because embossing would damage the internal devices.
- a three-year lithium battery
- complete compliance with the ISO standard package including 0.76 mm thickness [15].

A list of companies and products mentioned in this report appears in Appendix B.

1.2.2 Methods of reading

Cards may read in a variety of mechanical ways, which may be limited by the storage method utilized:

- Some, such as bank autoteller cards, are drawn into slots by a belt or wheel system, while a static magnetic head reads the magnetic stripe.
- The card may be clamped in position while a head, magnetic or optical, scans it.
- Some magnetic readers require the user to pass the card through a slot housing the head — these are called *swipe* readers.
- The card may have to be inserted into an electrical contact slot, as in the *Honeywell-Bull* and other smart cards.
- The card may merely be required to be brought within 20 cm of the reader, as in the *GEC* contactless smart card.

1.3 The case for Debit Cards

1.3.1 Some relevant Statistics

Currently coins and banknotes still account for the largest *number* of financial transactions, namely 75 percent, but these constitute transactions of relatively low *value* – about 3 percent [13, p 339].

There are about 30 million coin-operated pieces of equipment in the world of which most are drink-vending machines, followed by payphones and amusement machines.

There are about 50 000 coin payphones in South Africa.

Although cards are only used in a few percent of all financial transactions, they are becoming an acceptable alternative to other forms of money, as the following figures of interest show [13, p 341]:

- There are over 100 million magnetic stripe credit cards in the world.
- Over 320 million disposable magnetic stripe cards for payphones and railway are used each year in Japan. These are largely sold through automatic vending machines accepting banknotes and/or coins.

- Over 60 million holographic disposable phone cards have been sold to date in Europe.
- Tens of millions of memory-chip phone cards have been issued in France.

A recent market survey involving some 200 card-operated payphones conducted by *Telkor* and the South African Post and Telecommunications services has resulted in the sale of over 100 000 R10 and R5 cards, indicating public acceptance of the debit card as a means of payment, and sometimes generating more revenue per payphone than similar coin-operated units.

1.3.2 Advantages over a coin system

Advantages to the system administrators offered by a debit card system over coin validators include the following :

- the administering authority receives the money in advance of the service for which it is paying being delivered. This is a form of cheap credit.
- because the device does not contain coins which can be used elsewhere, there is no incentive for thieves. The damage to the device due to theft which can often be more than the value of the money stolen, as in payphones, is therefore curbed.
- the cost of the collection of coins is exchanged for the cost of administering the sale of cards. This is potentially a very favorable swap.
- a card system is more easily adaptable to changing value of currency than a coin validator because cards can be charged with values which are in non-monetary units.
- card-based systems are unaffected by the introduction of new coin types which a coin validator would have to validate and handle mechanically.

Users of the cards also benefit in that cards are a more compact form of money than coins.

An estimated cost per year of life [13, p 339] for some forms of payment are shown in table 1.1

payment type	life		cost per year	
coins	40	years	0.05	cents
magnetic stripe cards	3	years	5	cents
low-value banknotes	6	months	4	cents
supersmart cards	5	years	600	cents

Table 1.1: Cost per year of life for different forms of payment

The value of 40 years of life for a coin seems to be optimistic, and may only apply in economies boasting low inflation figures.

It can be seen that the average value of the transaction is important in choosing a payment technology.

The relative merits of debit or pre-payment cards and credit cards depend on the application, and in particular upon the necessity for on-line verification.

In the case of a stand-alone device such as a parking meter or vending machine it is not always feasible to make a phone call to verify the validity of the credit card presented if such a call has to be made via normal switching equipment.

In the case of a payphone, if a high-speed data link is available in the exchange, it may well be a practical option. An example of a payphone which offers both a credit card and a debit card facility is one developed by *Palindent Ltd.* of Israel [14]. Such a credit card reader is usually of the swipe type.

An alternative to on-line verification of credit cards in a payphone application is the storage of a 'hot-list' in the payphone itself. The credit card ID could be checked against this list and stolen or 'hot' cards can thus be denied. A log of transactions are stored in the host and reported regularly, for example daily, and the hot-list is updated via a call to a reporting centre. The disadvantage of this system is the memory requirement and the data base access software attendant to such storage.

1.4 System Requirements

User

The user of a card system has the following requirements:

- The card should be durable enough to withstand daily use and carrying in a pocket or wallet for a long enough period to be 'used up' for even lower than average usage. In the case of 'long life' cards, such as Smart Cards or otherwise rechargeable cards, it must withstand handling for such a time that the cost of the card can be amortized over its life. Some estimates of the typical life of cards of various types are [13, p 339]:
  - Magnetic stripe cards : 3 years
  - Chip-memory, contact smart, contactless smart cards : 3 years
  - Supersmart Cards : 5 years.
- The card must indicate visually and unambiguously, preferably also in a tactile fashion, for the benefit of blind people,
  - its value in monetary or other units, and
  - its intended mode of use, such as the direction and orientation of insertion.

Administration

The Administration requires the following from a debit-card reader system:

- each card should be cheap in relation to
  - if it is disposable, the monetary value of a new card or
  - if the card is re-chargeable, the average monetary value of transactions that will be made using the card during its life.
- Low maintenance. As one of the major advantages of a card system over a coin system is that the former does not require coin collection, the maintenance requirement of cleaning and battery replacement must be low enough so as not to destroy the cost advantag

Security

There should be a high probability that the system will detect and reject a fraudulent card.

A fraudulent card can be defined as either

- an authentic card of which the value has been increased by an unauthorized person, or
- a card manufactured to functionally resemble an authentic card.

If the card is disposable, it should be very difficult to increase the value remaining on a card, or to re-charge a card once it has expired.

If the card is rechargeable, it should be very difficult for an unauthorized person to charge the card.

Reliability

The demand for reliability means that there should be a high probability that the system will not reject an authentic card. It is of the utmost importance that

- the user does not lose his money by having his card rendered valueless, and
- valid cards are not rejected.

GEC [38] specify an overall card corruption rate of better than 1 in 1000, and a card misread rate of better than 4 in 100 insertions for their magnetic card reader.

Reliability also covers aspects such as resistance to dirt, dust and physical deformation of the card, user abuse of the device and temperature and humidity stability.

Chapter 2

An Overview of Cryptology

2.1 Introduction

The Greek word *krupt* [illegible] *hidden* — is the root word from which the following terms have been [illegible]rived [33].

Cryptology is the study of *cryptosystems*.

The two needs which cryptosystems can fulfill are

- *Secrecy*: When information is transmitted from one person to another, the two may desire that anyone else, overhearing the message, whether intentionally or not, should not be able to understand the message. This is also referred to as *privacy* or *confidentiality*.
- *Authentication*: when receiving a message, a person may want to be sure that that message was sent by a specific party, and not by someone else, even if that party later denies it; he would also like to be sure that the message sent by that party has not subsequently been altered.

Cryptology as a field covers two disciplines : *cryptography* is concerned with the *design* of *cryptosystems*, while *cryptanalysis* is the study of the *breaking* of cryptosystems. An extended list of possible motivations for breaking a cryptosystem is given in [19, p 516].

Cryptosystems are also called *ciphers*.

Cryptology today focuses on two main types of cryptosystem : classical *secret-key* systems, and the more recent *public-key* systems.

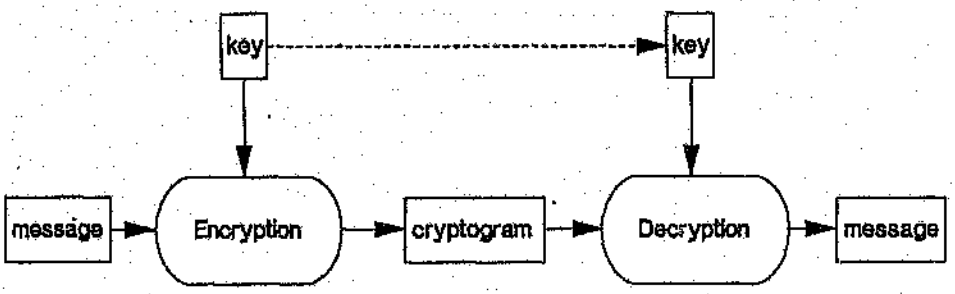

Figure 2.1: Encryption and Decryption

In secret-key systems a single piece of private information — the *key* — is shared by the originator and the recipient, and is used to *encrypt* the message into a *cryptogram* (figure 2.1). When the recipient receives the cryptogram, he uses the key to *decrypt* it into the original message. The implicit assumption, which is not always mathematically provable, is that it is (probably) impossible to perform these functions of encryption and decryption without knowledge of the secret key. Secret-key systems are discussed in section 2.2.

In public-key cryptography there are two keys [20, p 1], at least one of which it is computationally infeasible to discover from knowledge of the other. It is the fact that one of these need not be secret that gives public-key cryptography its name. The fact that secrecy and authentication are independent attributes of a cryptosystem was discovered with the advent of public-key systems. Public-key systems are discussed in section 2.3.

Cryptographic *algorithms* are methods of encryption or decryption.

Mathematically a cryptosystem consists of a message space M, a cryptogram space C, and a key space K (see figure 2.2) [21, p 27], [4, p 126]. Systems in which M and C are finite are known as *block ciphers*, and those for which M and C are infinite are called *stream ciphers*. In stream cipher systems the messages are referred to as *plaintext* and cryptograms as

ciphertext.

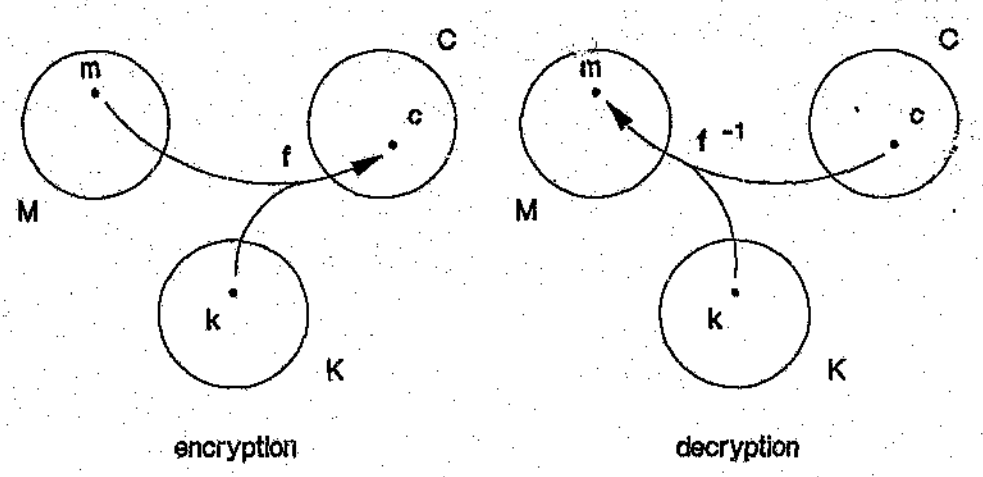

Figure 2.2: A Cryptosystem viewed mathematically

The process of encryption transforms a message $m \in M$ into a cryptogram $c \in C$ by the use of a key $k \in K$. The transformation is a function f which maps the product of M and K into C, i.e. $f : M \times K \longrightarrow C$, or $c = f(m, k)$. In order to ensure unique decryption, the function f must have a unique inverse such that $m = f^{-1}(c, k)$.

The business of proving the security of a cryptosystem mathematically is the domain of *provably secure* protocols.

Cryptanalysis can proceed on three levels [2, p 5]:

- *ciphertext only attack:* The cryptanalyst knows only a piece of ciphertext and often the context of the message.
- *known plaintext attack:* A piece of ciphertext with corresponding plaintext is known.

- *chosen plaintext attack:* The cryptanalyst chooses a piece of plaintext and has access to the corresponding ciphertext.

A cryptosystem is usually designed to withstand the most severe of these, a chosen plaintext attack.

2.2 Secret-key cryptosystems

2.2.1 Classical Systems

Substitution ciphers

In a substitution cipher [21, p 27] the plaintext alphabet is replaced with a secret alphabet. The key, known only to the originator and recipient, determines the alphabet.

- Mono-alphabetic ciphers (simple substitution) Every letter of the plaintext alphabet is replaced by another to generate the cryptogram.

  A simple example, the Caesar cipher, is due to Julius Caesar, who used the system of generating the cryptogram by rotating, by a secret number of letters — the key — cyclically through the alphabet each letter of the plaintext [2, p 7]. This is also known as an *additive* cipher [4, p 16]. For example, with a key of '4' (always Julius Caesar's choice), the plaintext-cryptogram pair in figure 2.3 could exist.

key:	4											
message:	j	u	l	i	u	s	c	r	y	p	t	o
cryptogram:	n	y	p	m	y	w	g	v	c	t	x	s

Figure 2.3: An example of encryption by the Caesar cipher

The simple substitution cipher has a small key space and is thus vulnerable to exhaustive key-search, i.e. trying all possible keys in turn until the message makes sense. In addition, the natural frequency of letters in the English or other language can be used to quickly break these simple cryptosystems [2, chapter 2].

- **Poly-alphabetic ciphers** Here, more than one secret alphabet is used [21, p 27].

  An example is the Vigenère cipher [2, p 9], [4, p 30] of 1586, in which a keyword is used repetitively, corresponding to using a number of Caesar ciphers periodically — each letter of the plaintext is encrypted with a different alphabet, and there are as many keys as letters in the key phrase. A *Vigenère square* (table 2.1) is used.

  If the letters 'a' to 'z' are assigned values 0 to 25, this table implements the function $c_i = m_i + k_i \bmod 26$ [21, p 27], where m_i is the i-th message character, k_i is the i-th character in the key, and c_i is the i-th character in the resulting cryptogram.

  This cipher, using the keyword 'code', would encrypt the message 'catchphrase' as shown in figure 2.4.

key:	c	o	d	e	c	o	d	e	c	o	d
message:	c	a	t	c	h	p	h	r	a	s	e
cryptogram:	e	o	w	g	j	d	k	v	c	g	h

Figure 2.4: An example of encryption by the Vigenère cipher

The Vigenère cipher does not suffer from a small key space, but can nevertheless be solved by an application of Kasisky techniques, an extension of the frequency analysis method [2, p 11], [21, p 27], [4, p 46].

The *Vernam* cipher is a Vigenère cipher with the keylength equal to the length of the message [2, p 14], and is also called the 'one-time pad'. It is unconditionally secure, and is the method used on the hot-line between Moscow and Washington.

Transposition cipher

This cipher reads the message characters row-wise and writes the cryptogram characters out column-wise, in an order determined by the key [2, p 15]. This does not alter the letter frequencies, and so it is usually used in combination with a cipher that does.

	a b c d e f g h i j k l m n o p q r s t u v w x y z
a	a b c d e f g h i j k l m n o p q r s t u v w x y z
b	b c d e f g h i j k l m n o p q r s t u v w x y z a
c	c d e f g h i j k l m n o p q r s t u v w x y z a b
d	d e f g h i j k l m n o p q r s t u v w x y z a b c
e	e f g h i j k l m n o p q r s t u v w x y z a b c d
f	f g h i j k l m n o p q r s t u v w x y z a b c d e
g	g h i j k l m n o p q r s t u v w x y z a b c d e f
h	h i j k l m n o p q r s t u v w x y z a b c d e f g
i	i j k l m n o p q r s t u v w x y z a b c d e f g h
j	j k l m n o p q r s t u v w x y z a b c d e f g h i
k	k l m n o p q r s t u v w x y z a b c d e f g h i j
l	l m n o p q r s t u v w x y z a b c d e f g h i j k
m	m n o p q r s t u v w x y z a b c d e f g h i j k l
n	n o p q r s t u v w x y z a b c d e f g h i j k l m
o	o p q r s t u v w x y z a b c d e f g h i j k l m n
p	p q r s t u v w x y z a b c d e f g h i j k l m n o
q	q r s t u v w x y z a b c d e f g h i j k l m n o p
r	r s t u v w x y z a b c d e f g h i j k l m n o p q
s	s t u v w x y z a b c d e f g h i j k l m n o p q r
t	t u v w x y z a b c d e f g h i j k l m n o p q r s
u	u v w x y z a b c d e f g h i j k l m n o p q r s t
v	v w x y z a b c d e f g h i j k l m n o p q r s t u
w	w x y z a b c d e f g h i j k l m n o p q r s t u v
x	x y z a b c d e f g h i j k l m n o p q r s t u v w
y	y z a b c d e f g h i j k l m n o p q r s t u v w x
z	z a b c d e f g h i j k l m n o p q r s t u v w x y

Table 2.1: Vigenère square

Shift register sequences

Shift registers [2, p 19], [21, p 28], [4, chapter 5] can be used to generate a pseudo-random sequence of bits. When used as the basis for an encryption algorithm, say by XORing the sequence with the message stream, a specific state of the shift register can be used as the secret key in the cryptosystem.

A South African general purpose encryption chip based on a 128-bit shift register is described in [42].

2.2.2 DES

DES (Data Encryption Standard) was adapted from an earlier system called LUCIFER, which had been developed by IBM for the American National Bureau of Standards [2, p 55]. It has been implemented on several chips — about three implementations are validated each year [3, p 552] — and is probably the most widely used secret-key cryptosystem in existence [7, p 19].

DES is a 64-bit block cipher. The algorithm has been published, e.g. [2, p 58], [3, p 268], [5, p 104], and is described in Appendix A.1.

The problems of key distribution encountered with secret-key systems led to the development of public key cryptography[2, p 63].

2.3 Public Key Cryptosystems

2.3.1 Introduction

The concept of public-key cryptographic systems was introduced in 1976 by the paper 'New Directions in Cryptography' by W. Diffie and M. E. Hellman.

The concept of a 'one-way function' was employed [10, p 371], and a 'trap-door one-way function' was a new concept introduced by this paper [6, p 543].

A trap-door one-way function (figure 2.5) is defined as a family of invertible functions f_z indexed by z, such that, given z, it is easy to find algorithms E_z and D_z that easily compute $f_z(x)$ and $f_z^{-1}(y)$ for all x and y in the domain and range, respectively, of f; but for virtually all z and for virtually all y in the range of f , it is computationally infeasible to compute $f_z^{-1}(y)$ even when one knows E_z.

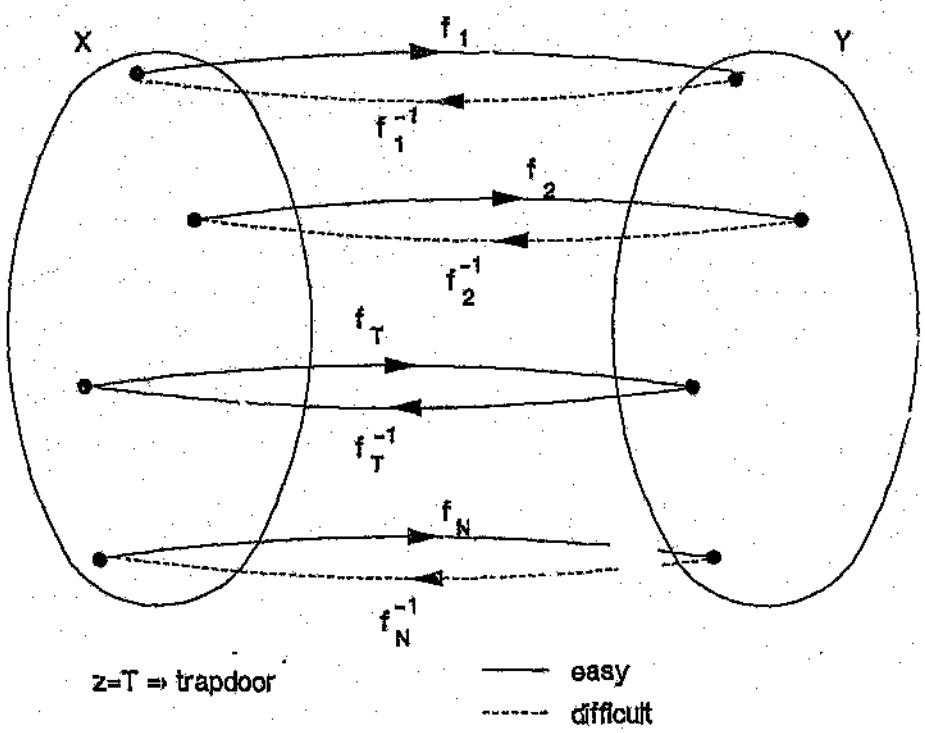

Figure 2.5: A Trapdoor one-way function

Various trapdoor one-way functions have been proposed as the basis for a public-key cryptosystem — based on prime numbers (as in the RSA system), knapsack problems, and Goppa codes (the McEliece system) [10, p 370-378], [2, p 93].

The public-key cryptosystem proposed by Diffie and Hellman provides a way for a secret message to be communicated between two users without having a secure channel for this communication in the following way [6, pp 543 – 544]:

- Like each user, Alice randomly chooses a value z_A of the index and keeps this as her private secret.
- She next forms the encryption algorithm E_{z_A} that she then publishes in a certified public directory.
- She also forms the decryption algorithm D_{z_A} that she keeps secret for her own use.
- If Bob wishes to send a secret message m to Alice, he fetches the algorithm E_{z_A} from the public directory. He uses this algorithm to compute the cryptogram $c = f_{z_A}(m)$ that he then sends to Alice.
- Alice uses her private algorithm D_{z_A} to compute $f_{z_A}^{-1}(c) = m$.

The same concepts are explained using different terminology in [10, p 370], [2, p 63] and [4, p 372].

The secret message thus transferred can be a key which can then be used in a conventional secret-key cryptosystem such as DES.

It should be noted, however, that this scheme does not obviate the need for authentication. The administrator of the public key library must be certain that only authorized users can place their key in the library, and each user must be certain that the keys of other users indeed came to him from the legitimate public key library [6, p 544].

To summarize [12, p 561], a public key cryptosystem has the following properties:

- The keys come in inverse pairs
- any message encrypted with one key can be decrypted with the other

- Given one member of the pair (the public key) it is infeasible to discover the other (the secret key).

Two implementations of public key systems, namely Exponential key exchange and the RSA system, are described in the next two sections.

2.3.2 Exponential Key Exchange

The scheme of exponential key exchange was one the first implementation of a public key cryptosystem, and is described in [12, p 563].

Exponential key exchange takes advantage of the relative ease with which it is possible to compute exponentials as compared with the difficulty of computing logarithms. This takes place in a Galois (finite) field $GF(q)$ (see [2, Appendix B] for an outline of the theory of finite fields).

$GF(q)$ has a prime number q of elements — the elements of the field are the numbers $\{0, 1, \ldots, q-1\}$, and arithmetic is done modulo q.

Let

$$Y = \alpha^X \bmod q, \qquad \text{for } 1 < X < q-1$$

where α is a fixed primitive element of $GF(q)$. This means that the powers of α produce all the nonzero elements $1, 2, \ldots, q-1$ of $GF(q)$. Then X is the logarithm of Y to the base α over $GF(q)$:

$$X = \log_\alpha Y, \qquad \text{for } 1 < X < q-1.$$

Calculating Y from X is easy, using the 'square and multiply' method [29, p 399]. For example,

$$\begin{aligned} \alpha^{21} &= \alpha^{16+4+1} \\ &= \left(\left((\alpha^2)^2\right)^2\right)^2 \times (\alpha^2)^2 \times \alpha. \end{aligned}$$

This will require at most $2 \times \log_2 q$ multiplications — work that the legitimate users of the key must do.

However, computing X from Y is typically far more difficult [12, p 562], and will require computation proportional to

$$L(q) = e^{\sqrt{\ln q \times \ln \ln q}}.$$

This is the 'discrete logarithm problem'.

If Alice wants to talk securely to Bob, she chooses a random number X_A uniformly from the integers $1, 2, \ldots, q$. She keeps X_A secret, and sends

$$Y_A = \alpha^{X_A} \bmod q$$

to Bob. Bob similarly chooses a random number X_B and sends

$$Y_B = \alpha^{X_B} \bmod q$$

to Alice.

The key that Alice and Bob will use in some system (probably a secret-key system) is now

$$K_{AB} = \alpha^{X_A X_B} \bmod q$$

Both Alice and Bob can compute this key. Alice computes

$$\begin{aligned} K_{AB} &= (Y_B)^{X_A} \bmod q \\ &= \left(\alpha^{X_B}\right)^{X_A} \bmod q \\ &= \alpha^{X_B X_A} \bmod q \end{aligned}$$

and Bob similarly computes

$$K_{AB} = (Y_A)^{X_B} \bmod q$$

If q is a prime about 1000 bits in length, then

- Alice must perform about 2000 multiplications to compute Y_A from X_A, or K_{AB} from Y_B and X_A.
- Bob must perform about 2000 multiplications to compute Y_B from X_B, or K_{AB} from Y_A and X_B.
- Anyone else (an intruder), not knowing either X_A or X_B, but only Y_A and Y_B, having intercepted these, must compute either of the logarithms $X_A = \log_\alpha Y_A$, or $X_B = \log_\alpha Y_B$. This currently demands more than 2^{100} (about 10^{30}) operations.

It should be noted, however, that it has not been proven mathematically that this problem of computing K_{AB} from Y_A and Y_B alone is equivalent to the discrete logarithm problem. Proving this equivalence is currently one of the major open problems in cryptography [12, p 563].

This scheme of exponential key exchange was proposed by W. Diffie in 'New Directions in Cryptography' in 1976. The RSA system expanded upon these ideas.

2.3.3 RSA

The RSA system was proposed by R. L. Rivest, A. Shamir, and L. Adleman of MIT in 1978 [6, p 544], [2, p 77]. It has become a very popular public-key system, and is under consideration for adoption as a standard by the ISO and CCITT [7, p 20], [34].

The security of the system is based on the assumption that the inversion of the function used, discrete exponentiation, is computationally as infeasible as factoring a large integer, the product of two integers, into its factors [6, p 545], [2, p 78], [8, p 42]. This assumption has not been proven for an exponent of greater than 2.

The RSA system is a block cipher in which the messages and cryptograms are integers between 0 and $n-1$ for some n [12, p 564]. It uses modular arithmetic (exponentiation) for encryption and decryption, as in the system of exponential key exchange, but does the arithmetic not over prime but over *bicomposite* numbers — the product of two primes.

The RSA system uses the fact that finding large, e.g. 200 digit, primes is comparatively computationally easy, but that factoring the product of two such primes appears to be computationally infeasible.

Use is made of a theorem by Leonard Euler [2, p 76, p 125], [4, p 194], [12, p 564]:

Let a and n be integers. Then

$$\gcd(a,n) = 1 \quad \Rightarrow \quad a^{\phi(n)} \equiv 1 \bmod n$$

where $\phi(n)$ is the *Euler totient function* defined by

$$\phi(n) = \|\{1 \leq i < n \mid \gcd(i,n) = 1\}\|^1$$

which gives the number of integers between 1 and n which are *coprime* to n i.e. they are relatively prime to n, and thus invertible modulo n.

For any prime p, $\phi(p) = (p-1)$.

For a bicomposite number $n = p \times q$ [6, p 381],

$$\phi(n) = (p-1) \times (q-1)$$

[1]Notation: $\|A\|$ denotes the cardinality (number of elements) of the set A.
$gcd(i,n)$ is the greatest common denominator of two integers i and n.
$\{a \mid b\}$ is the set of all a satisfying condition b.
Thus the right side of the definition counts the number of all i (with $1 \leq i < n$) such that $\gcd(i,n)$ is 1. For example, $\phi(10) = \|\{1,3,7,9\}\| = 4$.

Encryption Alice creates her secret and public keys by choosing at random two large primes, p_A and q_A, and multiplying them to obtain the bicomposite modulus $n_A = p_A \times q_A$. A suitable encryption exponent e_A is also chosen, such that $1 < e_A < \phi(n_A)$, and $\gcd(e_A, \phi(n_A)) = 1$. She then publishes the pair (e_A, n_A), but keeps the factors p_A and q_A secret — they no longer play a role, but cannot be made public.

Using Euclid's algorithm [29, pp 316–336], [2, pp 121–122] it is possible to find in less than $2 \cdot \log_2 \phi(n_A)$ operations another integer d_A such that

$$e_A \cdot d_A = 1 \bmod \phi(n_A).$$

This number d_A she keeps secret.

Anyone wishing to send Alice a message m can encrypt it by computing the cryptogram $m^{e_A} \bmod n_A$.

Decryption Bob, having similarly published his (e_B, n_B) pair, can decrypt a message m from the cryptogram $c = m^{e_B} \bmod n_B$ sent to him by Alice, or anyone else having access to the public directory, by computing c^{d_B} (only Bob knows d_B):

$$c^{d_B} = \left(m^{e_B}\right)^{d_B} = m^{1 \bmod \phi(n_B)} = m$$

Cryptanalysis Knowing the secret key d_B, a cryptanalyst can compute m from c in exactly the same way as Bob.

To find $\phi(n_B)$ from only the public n_B, however, he must perform about $O\left(e^{\sqrt{\ln n_B \ln\ln n_B}}\right)$ operations [2, p 78] to find the factors of n_B (Bob's p_B and q_B such that $p_B \times q_B = n_B$). This function grows according to table 2.2 [2, p 79]:

It should be noted that the difficulty of breaking this scheme has not been *proven* to be equivalent to that of factoring.

There is much confidence, however, that it is *much* more difficult to factor a large number than to multiply, because the problem of factoring is one which has been tackled by many great mathematicians without significant success. For numbers to be factored which are between 50 and 200 digits long, all of the best factoring algorithms today have running times which increase by a factor of ten for every 15 or so digits in the number. It would

digits in n	operations to factor n
50	1×10^{10}
75	9×10^{12}
100	2×10^{15}
125	3×10^{17}
150	3×10^{19}
175	2×10^{21}
200	1×10^{23}

Table 2.2: Number of operations needed for factorization

take a modern supercomputer about half a million years to factor a 200 digit number [6, p 545].

Hardware Implementation There is a trade-off between the speed of multiplication and the hardware required. Multiplication takes time proportional to the square lengths of the operands, $O(k^2)$. With dedicated serial/parallel hardware, this can be reduced to $O(k)$. The fastest implementations run at $O(logk)$, but the hardware required is then $O(k^2)$ [12, p 572].

Table 2.3 shows estimates of RSA encryption or decryption time for a 512-bit block [24, p 311 and p 322].

implementation	encryption time	
8-bit microprocessor	4	minutes
16-bit microprocessor	50	seconds
Texas TMS-32010 digital signal processor	2.6	seconds
Texas TMS-320C25 digital signal processor	<1	second
Motorola DSP 56200 digital signal processor	0.25	second
Inmos IMSA 100 Transputer and OCCAM	<0.25	second

Table 2.3: RSA encryption times for various hardware implementations

2.3.4 Digital Signatures

When, for every index z, the domain and range of a trapdoor one-way function f_z coincide, digital signatures can be created as follows: If user i wishes

to send a non-secret message s, the signature, to any or all users in the system, he computes and transmits the cryptogram $Y = f_{x_i}^{-1}(s)$. Every user, having access to E_{x_i}, can compute $f_{x_i}(Y) = s$, the 'signature'. However, only user i, knowing the trapdoor, can compute $Y = f_{x_i}^{-1}(s)$ in such a form that f_{x_i} is an intelligible and accepted signature [6, p 544], [9, p 46-47].

2.3.5 The Knapsack problem

A system based on the knapsack problem as the suggested trapdoor one-way function was proposed by Merkle and Hellman in 1978 [2, p 97].

The knapsack problem is this:

> given an integer S, and a sequence of n integers. Can a subset of the n integers be selected so that they add up to S?

In general, the knapsack problem is NP-hard (see [4, p 217] for discussion of NP-completeness). However, for certain sequences of n integers called superincreasing sequences [2, p 97], the problem is easy, and exhibits a trapdoor. The algorithm is described in [12, p 563].

Shamir broke the knapsack cryptosystem, not by solving the NP-hard problem, but by stripping off the disguise of the easy problem disguised as a NP-hard problem [27, p 580], [6, p 546].

2.3.6 Tamperfree Devices

Desmedt and Quisquater present in [25] the idea of a public-key cryptosystem which is based not on the computational complexity of a trapdoor one-way function, but on the tamperfreeness of some device, and present what they term an identity-based cryptosystem.

They argue that all practical cryptosystems make use of a tamperfree device in any case, and so propose a cryptosystem, the security of which rests on this tamperfreeness rather than on some computational problem.

2.4 Information Authentication

2.4.1 General Principles

Information Authentication is one of the two primary functions that cryptosystems can fulfill, the other being secrecy of information. In the application of a debit-card system, information authentication is the primary, perhaps only, concern. The emphasis is on being sure that a valid card has been presented, and not on keeping the information contained in the card secret.

Information authentication is, in simple terms, the determination by the authorized receiver(s), that a particular message was most probably sent by the authorized transmitter under the existing authentication protocol and that it has not subsequently been altered or substituted for [11, p 603].

An important implication of this statement is that in any particular authentication protocol, the receiver will accept as authentic only a fraction out of the total number of possible messages — the probability of an opponent 'guessing' a valid message should be low.

This, in turn, implies that authentication depends on the presence of redundant information either introduced deliberately, or else inherently present in the structure of the message, which will be recognizable to the receiver. The transmitter and receiver must therefore restrict their usage to only a small subset of all possible messages, which messages must contain the redundant information; other messages must be rejected because the transmitter would not have sent them [11, p 604].

Two approaches to authentication can be distinguished:

The first is a scheme whereby the redundant information is appended to, and functionally independent of, the message m that is to be authenticated. This necessitates that the extended message be encrypted as a whole, and that both the content of m and the redundant information which authenticates m be kept secret. If this were not done, the appended authenticator could be used to authenticate a fraudulent message.

The second scheme provides for the redundant, authenticating information to be a function of both a secret key and the message m which is to be authenticated, and therefore inseparable from m.

2.4.2 Merkle's Puzzles

A system proposed in 1974 [12, p 562] by R. Merkle involves the sending of a cryptographic key from one person to another by hiding it in a large collection of puzzles.

An intruder has to do about the square of the amount of work that a legitimate user has to do. This is not regarded as a large enough ratio.

2.4.3 Zero-knowledge proofs

Schemes for authentication have been proposed which are based on the idea of *non-transitive transfer of confidence* [35]. Using these protocols, Alice can convince Bob that she has the proof of some theorem, without transferring knowledge of the proof to Bob.

Similarly, it is possible for Alice to convince Bob that she knows a discrete logarithm, i.e. she knows an x such that $\alpha^x = \beta \bmod N$, without letting Bob know what x is [37].

In [36] protocols are proposed, using smart cards, to create 'unforgeable' ID cards.

Chapter 3

Magnetic Card System Implementation

3.1 Design Aims

A magnetic card reader is currently under development at *Telkor* R&D. The following targets were set for the system :

- The system response time to a card being inserted should be not more than about three to four seconds. The card phone market survey has shown that a response time in this order is acceptable to most users.
- The data capacity required of the card is largely determined by the application. In most of the applications under consideration the needs can be met by having
  - a card *value* field in number of units; a maximum value of 1000 (10 bits) seems to be acceptable, and
  - an *application ID* which indicates where a card should be treated as valid. This field can be used for limiting a card to be acceptable only at certain card readers, as would be useful in the photocopier-enabler application. The range of values of this field should be about 10 000 (14 bits).

- **Security** requirements are application-dependent, but on the whole it seems to be a firm specification that the effort required to
  - recharge used cards in large numbers, or
  - manufacture fraudulent cards in large numbers

  should be so great as to be non-cost-effective.
- The **cost** of the imported *GEC* card readers used in the *Telkor* market survey card phones was in the region of R 800, and this figure has been set as a target for the locally designed and produced one.
- Because of the motor required in the card reader mechanism, an internal battery is a likely requirement. **Power consumption** should therefore be as low as possible to prolong battery life before recharging.

3.2 Security Considerations

It should be noted at the outset that a card system is only as secure as the medium it employs. Only if the medium is difficult to copy, or tamperfree, can a cryptosystem provide additional security over and above that offered by the medium. Smart cards or optical cards, as far as the medium is concerned, are more secure than magnetic ones in sofar as it is more difficult for an opponent of the system to make or alter a smart card or a hologram than to make or alter a magnetic card.

For the purposes of encryption, the following assumptions have been made regarding the security of the system:

- the attacker can read data from a card that he has obtained legally
- he can write two-level data to the card, thus modifying the 'message' in the cryptosystem
- he cannot modify three-level data on the card
- the attacker does not have access to data in the card reader
- the attacker knows which encryption scheme is being used

Deterrents to fraud which can be included in the magnetic card system are the following:

- use of special high-coercivity magnetic tape which is less commonly used, and therefore less readily available
- use of special high-coercivity magnetic heads
- use of a three-level encoding technique to provide an authenticating 'signature' (see the 'medium key' description below)
- use of the 'magnetic hole' technique proposed below

Three-level medium key A simple three-level magnetic signature is implemented as follows:

An area or field on the card is encoded with a magnetic pattern in three-level format. The value of this field is unimportant. As the value on the card is decreased, a portion of the three-level medium-key is overwritten with a two-level or D.C.-value, one bit at a time. A check is always made on the length of the medium-key, and the allowed maximum value on the card is then limited to a value determined by the length of the three-level medium-key.

As an example, say 20 bits are reserved for the three-level medium-key, and say 10 two-level bits for a 'value' field. A new card contains a value of, say, 1000 units. As the card is used, the value remaining in number of units is decremented, and when it has been decremented by 1/20th of 1000 — i.e. it reaches 950 — one of the 20 bits of medium-key is destroyed. Subsequent reads will only now allow a maximum value of 950 units, so that a fraudulent value of higher than 950 will be rejected as invalid.

Assuming that the defrauder is unable to write three-level values successfully to the card, but can write two-level values of his choice, this scheme will force him to write a new value to the card every 1/20th, i.e. every 5 %, of its maximum value — every 50 units in the example above.

Magnetic hole technique In order to prevent the easy copying of cards, it may be feasible to adopt a scheme similar to the 'magnetic hole' method employed on software diskettes.

The scheme is this:

A magnetic hole is made in the magnetic material, i.e. a portion of the magnetic material is destroyed, so that it can no longer be magnetized. This hole should not be visible. The validating program, upon being presented with a card, decodes the position of the hole from the data on the card and then tries a write-read test in that location. If it is successful, it assumes that the card is fraudulent — it doesn't have the hole in the right place.

The position of the hole should be randomized at production and the physical burning of it should be automated.

The location of the hole resides, in encrypted form, on a field of the card. This field is a 'read-only' field.

In the case of the card, as opposed to the diskette, where the data is limited, the possible locations of the hole are likewise limited to the number of bits in the read-write field.

An implementation problem with this scheme is the difficulty of physically locating the hole position absolutely, without the benefit of a synchronization or 'clock' track, and in the presence of speed variation from one card mechanism to the next. It is necessary to be certain that adjacent bits will not be overwritten when the write-read test is done.

Another problem, as yet unaddressed, is that of creating these magnetic holes cost-effectively at production, and of keeping the hole invisible.

Keeping the number of transitions constant A scheme employed by the *GEC* magnetic card reader as a low-level security device is that of ensuring that the read-write area on the card always has the same number of peaks, when viewed as a trace on an oscilloscope, no matter what the value of the data in that area is [22, paragraph 3.4].

This means of course, that some redundant data is added to provide the extra transitions required to bring the total number of transitions up to a constant.

In the case of Manchester encoding being used (see figure 3.1), the number of transitions in a data field of d bits is

$$t_d = d + n_1$$

where n_1 is the number of 1's in the field.

A little examination will reveal that if no restriction is to be placed on the values that the data can assume, then the number of redundant bits required is equal to the data length:

Say the data field is d bits long and r redundant bits are to be added in order to keep the total number of transitions constant. The number of transitions in d is t_d, that in r is t_r, and the total number of transitions is $t = t_d + t_r$. Then the maximum number of transitions that can occur in the data field is

$$t_{d_{max}} = d + d = 2d$$

corresponding to a data field of all 1's. Similarly

$$t_{r_{max}} = r + r = 2r.$$

The minimum number of transitions (data all 0's) is

$$t_{d_{min}} = d.$$

Similarly

$$t_{r_{min}} = r.$$

The maximum number of 1's are added in r when $t_d = t_{d_{min}}$, which makes the total number of transitions

$$t_{min} = t_{d_{min}} + t_{r_{max}} = d + 2r$$

in this case, and when the data is all 1's, $t_d = t_{d_{max}}$, and the minimum number of 1's are added in r, so $t_r = t_{r_{min}}$, and

$$t_{max} = t_{d_{max}} + t_{r_{min}} = 2d + r.$$

But in order to keep the total number of transitions constant, we must have

$$t_{max} = t_{min},$$

i.e.

$$d + 2r = 2r + d,$$

which gives

$$r = d.$$

It is therefore feasible to add this feature of keeping the number of transitions constant if there is enough data capacity to merit a 100 % redundancy.

Encryption Scheme The primary aim of an encryption scheme for the debit card reader system is to devise a means for the card to convince the card reader that it is an authentic card.

For the magnetic card reader system, public key systems cannot be used because they require computation by the 'user' or card.

We are thus limited to secret-key systems. Of those systems discussed in chapter 2, the cryptographically most robust systems are DES and shift register sequences. The latter have the advantage that a cryptosystem can be designed to order, and to have a keyspace as large as one desires.

DES, on the other hand, is an existing field-proven system which thus has the advantage of having survived much intense scrutiny world wide. It also has a sufficiently large (2^{56}) key space for our application, and we can achieve a degree of customization by modifying the S-boxes (see Appendix A.1).

I have therefore decided to use the DES algorithm for encryption and decryption of the card data.

This may be done in a number of ways :

With the key in the card reader Because DES is a secret-key cryptosystem, this is where the key should theoretically reside, assuming that an attacker of the system does not have access to data in the card reader.

However, for some applications, such as that of a public phone, it is a requirement that any telephone card should be accepted by any telephone. Therefore either

- the key in all telephone card readers must be identical, in which case the key space is thus reduced to one! This means that once the key has been compromised (found or stolen), all telephone cards are liable to decipherment, or
- the telephone must manage a list of valid user or card ID's

This last problem exists even if a smart card is used. If each card is to be treated as a *unique* user, it is difficult to imagine how the DES scheme could be employed directly in the debit card application without the extensive memory required for storing the keys in the card reader or its host system. Also, because the key is only as safe as the reader, this is not a desirable setup for such applications as public telephones or parking meters. However,

a compromise may be possible if the number of keys is limited, although this would reduce the security.

With the key on the card This contradicts the use of DES as a secret-key system, but nevertheless complicates the task of the attacker.

One possible alteration that may be made to the system is to generate a modified set of tables for the DES algorithm, and keep these secret within the card reader. These may be kept in battery-backed RAM — or other protected but erasable memory — and erased when the card reader is tampered with, as an additional secrecy device. The disadvantage of this scheme is that the benefit of the confidence in the security of the system arising from extended use in practice by many agencies all over the world is lost. However, if the tables are randomized, this does not seem to be likely to be a problem, and the advantage gained is that our cryptosystem will be intact even if *DES* were compromised.

Using a split key In practice, it may be feasible to use the DES algorithm with a key partially stored in the card reader and partially on the card. Authentication can be achieved by accepting as valid only a small subset of all possible keys.

It may also be possible to use the DES algorithm twice; once using a full 64-bit key on the card, and a second time using a full key residing in the card reader. The practicality of this alternative will depend on the execution time of the algorithm.

The DES algorithm is described in Appendix A.1

In order to gain familiarity with DES, a PC-simulation of the encryption algorithm was written in Turbo Pascal. The source code for this simulation is given in Appendix A.2.

3.3 System Description

The system works as follows:

Plastic cards of ISO credit card size, but of a reduced thickness of 0.45 mm have been manufactured by *Brown, Davis & McCorquodale* for this purpose.

The card has a single, central, 5 mm magnetic stripe, heat-pressed to the surface. The high-coercivity magnetic tape was developed by the *Kurz*

company in Germany. A magnetic head obtained from *Magnetic Components* in Chertsey, England is used [32]. It has a read width of 1.1 mm and a write width of 4.7 mm, which obviates the need for very accurate physical location in the reader mechanism. A 12 V D.C. motor is used to drive the card back and forth past the head by means of a rubber wheel or belt.

The reason for the reduced thickness of the card is that in a new card mechanism, currently being prototyped by Jack Cook in *Telkor* R&D, the card will be bent around a drum by a drive belt to reduce longitudinal travel required to parade the card past the head, making for a more compact unit.

A cheap magnetic head will be positioned outside the card mechanism, in line with the magnetic stripe. When the card is presented to the card slot, the first few bits on the magnetic stripe will provide a signal from this head which will be used to open a mechanical shutter, allowing the rest of the card to be drawn in.

Low-coercivity forgeries will be erased upon entry.

The strip has been centralized for two reasons:

- the ANSI standard [30] for financial transaction cards specifies the magnetic material to be on one side of the card, 5.54 mm maximum from the edge. Because the security requirements of the debit card system have necessitated the use of a high-coercivity magnetic medium, the system cannot conform to the magnetic standards of ISO and ANSI. It is therefore considered better to not allow standard credit-card-type cards into the card reader mechanism. Placing the shutter head in the center of the card slot will prevent credit card type cards from opening the shutter.

- it is conceivable that in some applications the function of the card may be varied in response to the direction of insertion. For instance, on their parking meter card *GEC* provide a 'check' function, merely displaying the card value, when inserted in one direction, and a 'debit' function, decreasing the card value, when inserted in the other direction. This is possible — using one head — only when the card stripe is centralized.

An Intel 80C32 microcontroller running at 3.6 MHz is used to control reading from and writing to the head, reading opto-sensors for sensing card position, and controlling the motor to drive the card.

The power supply and head and motor control circuitry has been designed by Dawood Kadwa, and the control software written by Joanne Carey, under my supervision.

Two methods of magnetic encoding are used:

Two-level encoding The two-level encoding scheme used is the Manchester encoding method [31, pp 279–2[illegible], which works as follows (figure 3.1):

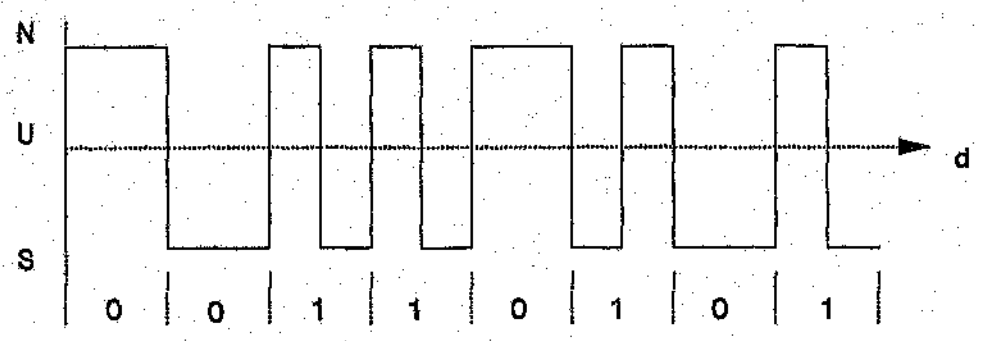

Figure 3.1: Manchester encoding

- each bit occupies a fixed length along the distance d on the card.
- at the end of each bit, a transition occurs from North to South or South to North
- for a bit value of '1', an additional transition occurs in the middle of the bit.

This code, also called 'DF' (for 'Double Frequency'), PE-M ('Phase-Encoding Mark'), or 'Bi-ϕ-M' ('Bi-phase, Mark') [31, p 282] is self-clocking. Although it is a 'double-frequency' code, requiring double bandwidth, the software seems to be able to keep up with relative ease at a clock speed of 3.6 MHz. In addition, it does not require DC response of the read circuitry, a desirable attribute at small signal levels.

Three-level encoding This code has been modified from the Manchester code by Dawood Kadwa to utilize three magnetic levels as follows (figure 3.2):

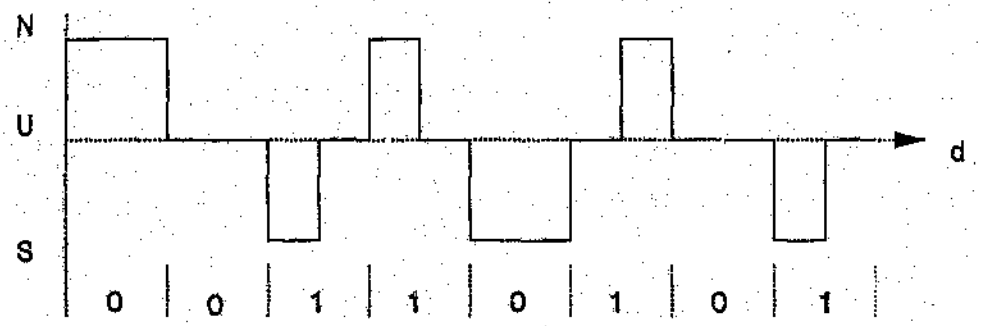

Figure 3.2: Three-level encoding

- Three levels of magnetization are possible : Un-magnetized virgin tape (U), North (N) and South (S).
- The term 'transition' in this case means a change from N to U, U to N, S to U, or U to S. Changes are not made from S to N or N to S.
- Transitions occur in a cyclical fashion, i.e. N-U-S-U.
- As in the case of manchester encoding, transitions occur at the end of each bit, and in addition in the middle of a bit if its value is '1'.

3.4 Results

3.4.1 System requirements

The requirements of the system administration as regards durability and reliability mentioned in chapter 1 have not yet provably been met, as the new card reader mechanism is still under development, and a field trial is only envisaged later this year.

However, sample cards have been read and written as many as 10 000 times, so no problems are anticipated with the cards themselves.

Power consumption figures for the card reader electronic circuitry indicate that the motor will be a major consumer. Such figures will become available upon completion of the new card reader mechanism.

No major format variation has been entered upon as far as the user is concerned, so that the positive results obtained from the card phone market survey as regards user acceptance of the card as a form of payment are expected to apply equally to the new system.

Initial estimates of the cost of the magnetic cards range from 50 cents to 80 cents. This is likely to be acceptable for all of the applications under consideration.

Initial estimates of the cost of the card reader indicate a figure of around R 700, which is higher than anticipated, but still cheaper than any known alternative, and meets the design aim.

The system response time, including decryption, upon the insertion of a card, is about three seconds, of which the DES algorithm takes one second. This meets the design aim.

It has been found empirically that a data density of about 23 bits per cm is achievable, giving about 200 bits over the length of the card. Data in both two-level and three-level formats has been successfully written and read. This data capacity achieved exceeds the read-write requirement for all applications under consideration as well as providing enough capacity to implement the security devices mentioned earlier.

3.4.2 Security

Cryptosystem

Cryptographically, and under the assumptions mentioned in section 3.2, 'breaking' the system in our implementation would be that an attacker can change the value of his card. In order to do this, he must know what data to write in the read-write field containing the value of the card in encrypted form. The key is secret in the card reader.

The following scenarios confront the would-be cracker of the system:

- *ciphertext only attack*: The cryptanalyst knows only a piece of ciphertext — only the data written on a read-write field on the card is known,

not knowing what it represents.

- *known plaintext attack:* A piece of ciphertext with corresponding plaintext is known — this is easily achieved by knowing how a specific value, say the value of a new card, is encoded.
- *chosen plaintext attack:* The cryptanalyst chooses a piece of plaintext and has access to the corresponding ciphertext — he uses a specific number of units of a card and then studies the resulting encoded card value.

A chosen plaintext attack is thus relatively easy to attempt.

However, there is currently no known method of breaking the DES algorithm [4, pp 283 – 284]. Assuming that this remains to be the case, an exhaustive search is the only alternative. This will have to be done over 2^{56} possible keys. Even at the rate of one key every nanosecond this will take longer than two years.

Magnetic Hole

The magnetic hole scheme is intended to impede the copying of cards. In order to defeat it, the copyist must find out where the hole is on one card and produce a hole in the copied card in the same place.

Medium Key

A small loophole exists for the attacker who can write two-level data of his choice to the card. He can

- buy a new card
- record the data in the read-write field somewhere
- use the card for so many units that it just does not reach the first erasure of a medium key bit. At this point the three-level data on the card is the same as that of a new card.
- write the original, stored value of the read-write field back to the card.

However, this can become very tedious if the number of bits in the medium field is sufficiently high. For example, with 20 bits here, he has to re-write the card every 5% of its full value.

Chapter 4

Smart Card System

In order to justify the cost of a smart debit card, the cards will need to be rechargeable. This presents the added security requirements of the recharging method and machinery, and this aspect will have to be considered in the design of the system as a whole.

If a smart card is used, the security of the system is potentially greatly enhanced. Security measures which can be employed are the following:

- Smart cards are technologically much more difficult to copy or to tamper with than magnetic cards.
- A unique secret key for every card which really *is* secret can be stored in the card by blowing fuses upon initializing the card, preventing access from outside.
- Because the card has processing ability, algorithms can be computed inside the card. These may be secret-key as well as public-key algorithms. However, the computation times may be prohibitively long for a scheme such as RSA (table 2.3) for current practical microprocessors, and dedicated encryption hardware will most likely have to be included on the card.

A possible authenticating procedure in a debit card application using RSA public key methods (see section 2.3.3) is as follows [11, p 608]:

- The encryption algorithm and key pair $(e; n)$ are kept secret, while the decryption algorithm and pair $(d; n)$ are made public.

- Messages are integers m, with the authenticating redundant information of say fifty trailing zero bits. The card sends the cryptogram $c = m^e \bmod n$ to the card reader.
- The card reader will accept as valid any message $m = c^d \bmod n$ which has fifty trailing zeroes.
- The probability of guessing a valid message is 2^{-50}, or about 10^{-15}.

Despite much work in the field, this scheme is as yet not provably secure.

The practical implementation of such a cryptosystem will require much thought to determine how many users (cards) can be accommodated with limited memory available in the host, and what the messages are to be.

A real-time variable such as the time and date should be sent to the card and used in the algorithm to prevent the presentation of recorded responses to the card reader.

Chapter 5

Conclusion

Debit cards provide an attractive solution to the demand for flexibility in unattended fee-collection systems and circumvent the administrative problems associated with traditional coin-based systems. Primary among the advantages are the obviation of the need to collect coins from each installation, the fact that there is no money to be stolen, and that the seller of the service receives his money in advance.

A literature survey was done in order to gain an overview of which card formats were in use and could be applied to the debit card implementation. It was found that the most popular types were optical or hologram cards, magnetic cards, and smart cards. Of these, the most accessible technology was that of magnetic cards, and in order to enhance the security of the system it was decided that a high-coercivity magnetic strip would be used because this provided greater protection against accidental or fraudulent alteration of the card information.

A study was made of cryptological techniques, from traditional manual systems to modern public-key methods, in order to ascertain how these could be applied to debit cards. The performance of available low-cost microprocessors make implementations of cryptosystems of the calibre of DES not only feasible, but *necessary*, because attackers have access to the same computational power. Public-key systems such as RSA may require dedicated encryption chips, but are likely to become even more popular as such hardware becomes affordable.

A magnetic card reader system was developed successfully to the prototype stage using the security devices of multi-level encoding and the DES

encryption scheme.

Although the cryptological aspect of the magnetic card implementation is satisfactory under the assumptions made, any attack on the system is not likely to be cryptanalytical in nature. Rather, it is likely that the attack will be an attempt to forge, copy, or alter the magnetic medium which carries the card information.

A smart card, containing as it does a silicon chip, provides much greater medium security, and has the added advantages of much greater data capacity and computational ability which can flexibly utilize the benefits of the best cryptographic schemes for its authentication function.

At present only the cost of a smart card system places it at a disadvantage over other card formats.

Appendix A

DES

A.1 Description of the Algorithm

The DES algorithm [2, p 57] is shown in figure A.1. It operates as follows:

- The 64 bits of the message are permuted by the Initial Permutation IP (table A.1)

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

Table A.1: Initial permutation IP

  Thus the 58-th bit of the message goes to position 1, the 50-th bit goes to position 2, and so on. The left most 32 bits go to L_0, while the rightmost 32 go to R_0

- These two halves then go through 16 rounds of manipulation, at the end of which the result is again permuted by an Inverse Initial Permutation IP^{-1} (table A.2).

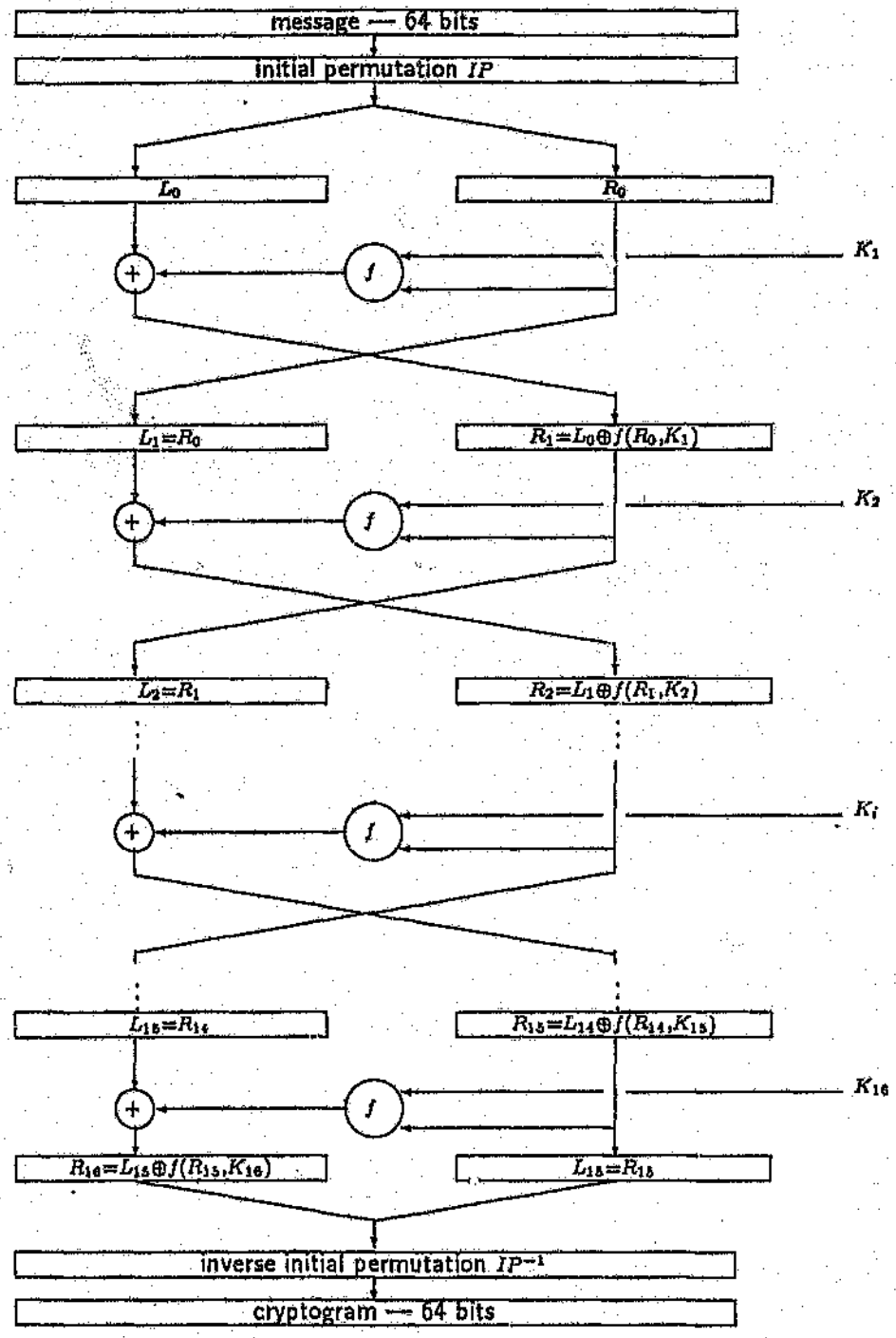

Figure A.1: The DES algorithm

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

Table A.2: Inverse Initial permutation IP^{-1}

- At any of the 16 rounds, lets call the halves L_i and R_i. Then

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

and

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

- The function f is computed as shown in figure A.2.

  The 32-bit vector of R_{i-1} is expanded to 48 bits by the bit selection table E (table A.3).

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Table A.3: Bit Selection table E

This sends bit 32 to position 1 and position 47, bit 1 goes to position 2, and so on. The 48 bits are processed by selection functions called S-boxes. The first 6 bits go to S-box S_1, the next 6 bits S-box S_2, etc. Each S-box has 4 output bits. The 32 bits resulting from the S-boxes are then permuted according to the permutation P (table A.4).

This sends bit 16 to position 1, bit 7 to position 2, etc.

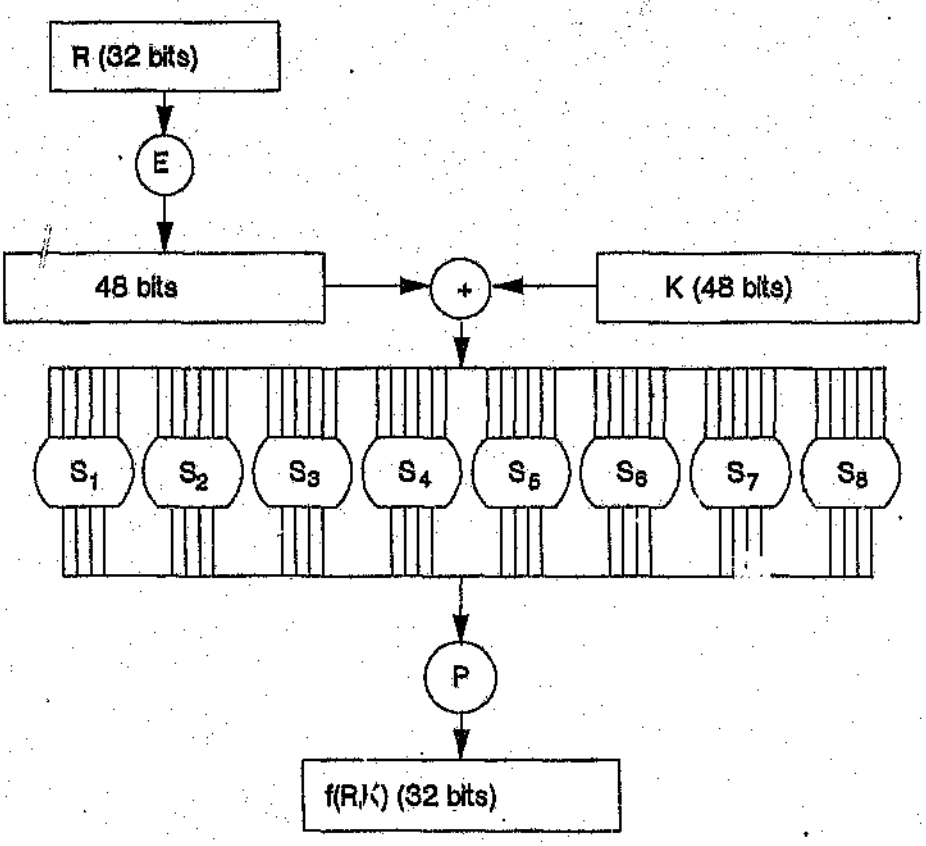

Figure A.2: Calculation of the cipher function f

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Table A.4: Permutation P

- The S-box structure is shown in table A.5.

  Each S-box takes as input a 6-bit number, say B. The first and last bits of B form a binary number in the range 00 to 11 binary, or 0 to 3 decimal — call this number i. The middle four bits of B form a number in the range 0000 to 1111 binary, or 0 to 15 decimal. Call that number j. Look up in the S-box in row i and column j a four-bit number. This is the output of the S-box. For example, in S-box S_1, the input 111000 tells us to look up in row 10 (2 decimal) and column 1100 (11 decimal) the number 7, which yields 0111 as the output.

  The addition in figure A.2 is XORing. The result $f(R_{i-1}, K_i)$ is a binary vector of length 32.

- The sixteen 48-bit vectors K_i in figure A.1 are derived from the Key Schedule calculation (figure A.3).

  The key is divided into eight blocks of eight bits, including a parity bit, each. The 56 non-redundant bits are selected in an order indicated by the permuted choice $PC-1$ (table A.6).

  This puts bit 57 in position 1, bit 49 in position 41, etc. The leftmost 28 of these permuted bits are fed to C_0, and the rightmost 28 to D_0. In the subsequent 16 rounds, the C_i and D_i are shifted left by either 1 or 2 positions, as specified by the left shift table LS (table A.7).

  Permuted choice $PC-2$ (table A.8) selects for each of the sixteen rounds, a 48-bit vector K_i from C_i and D_i to be used in the cipher function f.

S_1

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

S_2

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

S_3

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

S_4

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

S_5

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

S_6

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

S_7

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

S_8

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

Table A.5: S-boxes (selection functions)

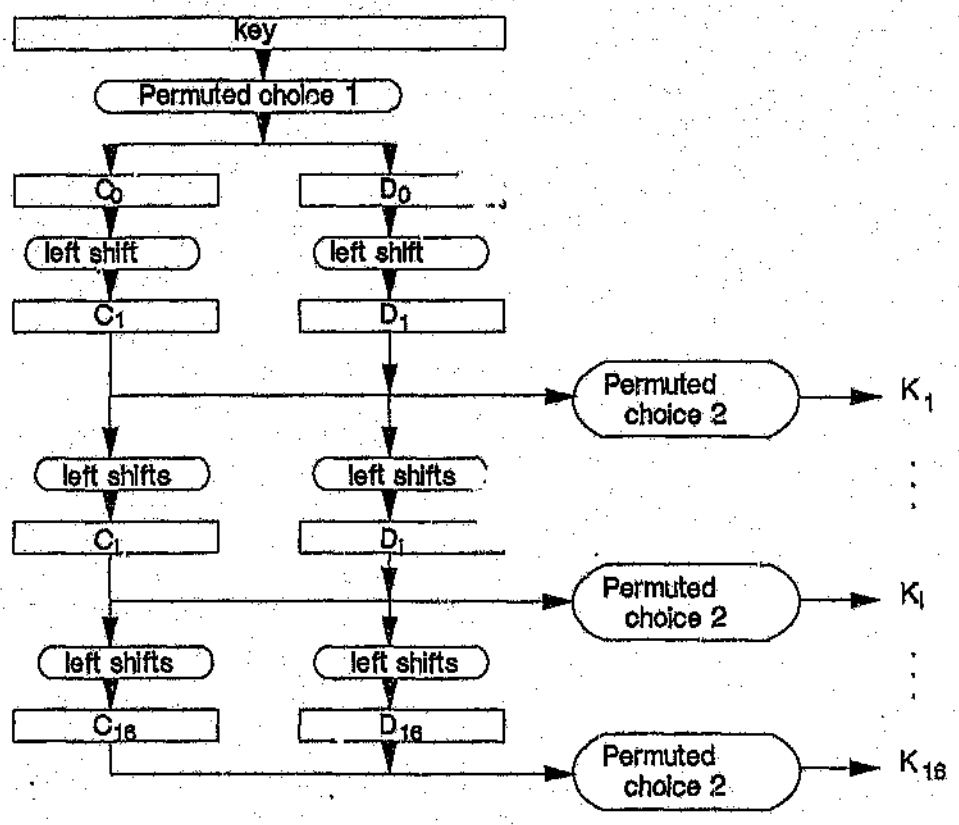

Figure A.3: Key Schedule calculation

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

Table A.6: Permuted choice $PC-1$

round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Table A.7: Left Shift table LS

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

Table A.8: Permuted choice $PC-2$

- The inverse function, decryption, is performed by following figure A.1 from the bottom to the top, where

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

and

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

A.2 PC-simulation in PASCAL

```
{ ------------------------------------------------------------------- }
{   DES unit ... Kobus Botha  January - May 1990                      }
{ ------------------------------------------------------------------- }

unit DES;

interface

type
    msg_4   = array[0..3]  of byte;
    msg_6   = array[0..5]  of byte;
    msg_28  = array[0..27] of byte;
    msg_32  = array[0..31] of byte;
    msg_48  = array[0..47] of byte;
    msg_56  = array[0..55] of byte;
    msg_64  = array[0..63] of byte;msg_array     =[0..7] of byte;
    DES_rec             = record
                            msg     :_64;
                            l_msg   : array[0..16] of msg_32;
                            r_msg   : array[0..16] of msg_32;
                            key     : msg_64;
                            c_key   : array[0..16] of msg_28;
                            d_key   : array[0..16] of msg_28;
                            K       : array[1..16] of msg_48;
                            cryptogram : msg_64;
                          end;


procedure EnDES(var des : DES_rec);
procedure DeDES(cryptogram_in : byte_msg_array; key : byte_msg_array;
                var msg_out : byte_msg_array);

procedure IP(in_msg :_64; var l_out, r_out : msg_32);
procedure InverseIP(l_in, r_in : msg_32; var out_msg : msg_64);
procedure MsgBytesToBits(msg_in : byte_msg_array;
                         var msg_out :  sg_64);
procedure MsgBitsToBytes(msg_in : msg_64;
                         var msg_out : byte_msg_array);
procedure PC_1(key_in : msg_64; var c_out : msg_28; var d_out : msg_28);
procedure LeftShift( eration : byte; c_in, d_in : msg_28;
                                 var c_out, d_out : msg_28);
procedure PC_2(c_in, d_in : msg_28; var key_out : msg_48);
procedure FunctionP(r_in : msg_32; key_in : msg_48; var f_out : msg_32);
procedure XOR32(msg_1, msg_2 : msg_32; var msg_out : msg_32);
```

```
{ ------------------------------------------------------------------ }
implementation

const
    IP_TABLE : array[0..63] of byte =
               (   58, 50, 42, 34, 26, 18, 10, 2,
                   60, 52, 44, 36, 28, 20, 12, 4,
                   62, 54, 46, 38, 30, 22, 14, 6,
                   64, 56, 48, 40, 32, 24, 16, 8,
                   57, 49, 41, 33, 25, 17,  9, 1,
                   59, 51, 43, 35, 27, 19, 11, 3,
                   61, 53, 45, 37, 29, 21, 13, 5,
                   63, 55, 47, 39, 31, 23, 15, 7);

    INVERSE_IP_TABLE : array[0..63] of byte =
               (   40,  8, 48, 16, 56, 24, 64, 32,
                   39,  7, 47, 15, 55, 23, 63, 31,
                   38,  6, 46, 14, 54, 22, 62, 30,
                   37,  5, 45, 13, 53, 21, 61, 29,
                   36,  4, 44, 12, 52, 20, 60, 28,
                   35,  3, 43, 11, 51, 19, 59, 27,
                   34,  2, 42, 10, 50, 18, 58, 26,
                   33,  1, 41,  9, 49, 17, 57, 25);

    E_BIT_SEL_TABLE  : array[0..47] of byte =
               (   32,  1,  2,  3,  4,  5,
                    4,  5,  6,  7,  8,  9,
                    8,  9, 10, 11, 12, 13,
                   12, 13, 14, 15, 16, 17,
                   16, 17, 18, 19, 20, 21,
                   20, 21, 22, 23, 24, 25,
                   24, 25, 26, 27, 28, 29,
                   28, 29, 30, 31, 32,  1);

    PC_1_TABLE  : array[0..55] of byte =
               (   57, 49, 41, 33, 25, 17,  9,
                    1, 58, 50, 42, 34, 26, 18,
                   10,  2, 59, 51, 43, 35, 27,
                   19, 11,  3, 60, 52, 44, 36,
                   63, 55, 47, 39, 31, 23, 15,
                    7, 62, 54, 46, 38, 30, 22,
                   14,  6, 61, 53, 45, 37, 29,
                   21, 13,  5, 28, 20, 12,  4);

    LS_TABLE  : array[0..15] of byte =
               (    1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1);
```

```
PC_2_TABLE  : array[0..47] of byte =
            (   14, 17, 11, 24,  1,  5,
                 3, 28, 15,  6, 21, 10,
                23, 19, 12,  4, 26,  8,
                16,  7, 27, 20, 13,  2,
                41, 52, 31, 37, 47, 55,
                30, 40, 51, 45, 33, 48,
                44, 49, 39, 56, 34, 53,
                46, 42, 50, 36, 29, 32);

S_TABLE : array[0..7, 0..3, 0..15] of byte =
{ S1 }(((14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7),
        ( 0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8),
        ( 4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0),
        (15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13)),

{ S2 } ((15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10),
        ( 3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5),
        ( 0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15),
        (13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9)),

{ S3 } ((10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8),
        (13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1),
        (13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7),
        ( 1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12)),

{ S4 } (( 7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15),
        (13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9),
        (10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4),
        ( 3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14)),

{ S5 } (( 2, 12, 14,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9),
        (14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6),
        ( 4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14),
        (11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3)),

{ S6 } ((12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11),
        (10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8),
        ( 9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6),
        ( 4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13)),

{ S7 } (( 4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1),
        (13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6),
        ( 1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2),
        ( 6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12)),

{ S8 } ((13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7),
        ( 1, 15, 13,  8, 10,  3,  7,  4, 11,  5,  6, 11,  0, 14,  9,  2),
        ( 7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8),
        ( 2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11)));
```

```
{ ------------------------------------------------------------------ }
procedure MsgBitsToBytes(msg_in : msg_64;
                         var msg_out : byte_msg_array);
var
    i, j, k : integer;
begin
    j := 0; k := 0;
    for i := 0 to 63 do
        begin
            if msg_in[i] = 1 then
                msg_out[j] := msg_out[j] OR  ($80 shr k)
            else
                msg_out[j] := msg_out[j] AND NOT($80 shr k);
            inc(k);
            if k = 8 then
                begin
                    k := 0;
                    inc(j);
                end;
        end;
end; { MsgBitsToBytes }
{ ------------------------------------------------------------------ }


{ ------------------------------------------------------------------ }
procedure MsgBytesToBits(msg_in : byte_msg_array;
                         var msg_out : msg_64);
var
    i, j, k : integer;
begin
    j := 0;
    for i := 0 to 7 do
        begin
            for k := 0 to 7 do
            begin
                if (msg_in[i] AND ($80 shr k)) <> 0 then
                    msg_out[j] := 1
                else
                    msg_out[j] := 0;
                inc(j);
            end;
        end;
end; { MsgBytesToBits }
{ ------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure XOR32(msg_1, msg_2 : msg_32; var msg_out : msg_32);
var
    i : integer;
begin
    for i := 0 to 31 do
        msg_out[i] := msg_1[i] XOR msg_2[i];
end; { XOR32 }
{ ------------------------------------------------------------------------ }


{ ------------------------------------------------------------------------ }
procedure PC_2(c_in, d_in : msg_28; var key_out : msg_48);
var
    i_          : integer;
    ln          : byte;
    temp_56     : msg_56;
    PC_2_index  : byte;
begin
    for i_ := 0 to 27 do
        begin
            temp_56[i_]      := c_in[i_];
            temp_56[i_ + 28] := d_in[i_];
        end;

    for i_ := 0 to 47 do
        begin
            PC_2_index := PC_2_TABLE[i_] - 1;
            key_out[i_] := temp_56[PC_2_index];
        end;
end; { PC_2 }
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure SBox(s_box_number : byte; b_in : msg_6; var b_out : msg_4);
var
    i       : byte;
    j       : byte;
    table_entry     : byte;
begin
    i := ((b_in[0] AND $01) shl 1) OR (b_in[5] AND $01);
    j :=    ((b_in[1] AND $01) shl 3)
        OR  ((b_in[2] AND $01) shl 2)
        OR  ((b_in[3] AND $01) shl 1)
        OR  ((b_in[4] AND $01));
    table_entry := (S_TABLE[s_box_number, i, j]);
    if (table_entry > $0F) then
        begin
            write(' S-box error ! ');
            halt(1);
        end;
    b_out[0] := (table_entry AND $08) shr 3;
    b_out[1] := (table_entry AND $04) shr 2;
    b_out[2] := (table_entry AND $02) shr 1;
    b_out[3] := table_entry AND $01;
end; { SBox }
{ ------------------------------------------------------------------------ }


{ ------------------------------------------------------------------------ }
procedure BitSelectE(r_msg_in : msg_32; var out_msg : msg_48);
var
    msg_index   : integer;
    E_index     : integer;
begin
    for msg_index := 0 to 47 do
        begin
            E_index := E_BIT_SEL_TABLE[msg_index] - 1;
            out_msg[msg_index] := r_msg_in[E_index];
        end;
end; { BitSelectE }
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure FunctionF(r_in : msg_32; key_in : msg_48; var f_out : msg_32);
var
    temp_48     : msg_48;
    i,j         : byte;
    s_in_mark   : byte;
    f_out_mark  : byte;
    b_6         : msg_8;
    b_4         : msg_4;
begin
    BitSelectE(r_in, temp_48);
    for i := 0 to 47 do
        begin
            temp_48[i] := temp_48[i] XOR key_in[i];
        end;

    s_in_mark := 0;
    f_out_mark := 0;
    for i := 0 to 7 do
        begin
            s_in_mark := i * 6;
            for j := 0 to 5 do
                b_6[j] := temp_48[s_in_mark + j];
            SBox(i, b_6, b_4);
            for j := 0 to 3 do
                f_out[f_out_mark + j] := b_4[j];
            inc(f_out_mark, 4);
        end;
end; { FunctionF }
{ ------------------------------------------------------------------------ }


{ ------------------------------------------------------------------------ }
procedure LeftShift(iteration : byte; c_in, d_in : msg_28;
                                var c_out, d_out : msg_28);
var
    i       : integer;
    ls      : byte;
begin
    ls := LS_TABLE[iteration];
    for i := 0 to 27 do
        begin
            c_out[i] := c_in[(i + ls) mod 28];
            d_out[i] := d_in[(i + ls) mod 28];
        end;
end; { LeftShift }
{ ------------------------------------------------------------------------ }
```

```
{ ---------------------------------------------------------------------- }
procedure PC_1(key_in : msg_64; var c_out : msg_28; var d_out : msg_28);
var
    i           : integer;
    PC_1_index  : byte;
begin
    for i := 0 to 27 do
        begin
            PC_1_index := PC_1_TABLE[i] - 1;
            c_out[i] := key_in[PC_1_index];
        end;
    for i := 28 to 55 do
        begin
            PC_1_index := PC_1_TABLE[i] - 1;
            d_out[i-28] := key_in[PC_1_index];
        end;
end; { PC_1 }
{ ---------------------------------------------------------------------- }


{ ---------------------------------------------------------------------- }
procedure IP(in_msg : msg_64; var l_out, r_out : msg_32);
var
    i           : integer;
    IP_index    : byte;
begin
    for i := 0 to 31 do
        begin
            IP_index := IP_TABLE[i] - 1;
            l_out[i] := in_msg[IP_index];
        end;
    for i := 32 to 63 do
        begin
            IP_index := IP_TABLE[i] - 1;
            r_out[i-32] := in_msg[IP_index];
        end;
end; { IP }
{ ---------------------------------------------------------------------- }
```

```
{ ------------------------------------------------------------------------ }
procedure InverseIP(l_in, r_in : msg_32; var out_msg : msg_64);
var
    i          : integer;
    IP_index   : byte;
begin
    for i := 0 to 63 do
        begin
            IP_index := INVERSE_IP_TABLE[i] + 1;
            if IP_index < 32 then
                out_msg[i] := l_in[IP_index]
            else
                out_msg[i] := r_in[IP_index-32];
        end;
end; { InverseIP }
{ ------------------------------------------------------------------------ }


{ ------------------------------------------------------------------------ }
procedure EnDES(var des : DES_rec);
var
    i       : byte;
    temp_32 : msg_32;
begin
    with des do
    begin
        IP(msg, l_msg[0], r_msg[0]);
        PC_1(key, c_key[0], d_key[0]);
        { -------------------- }
        for i := 0 to 15 do
            begin
                { -----, Key Select ... }
                LeftShift(i, c_key[i], d_key[i], c_key[i+1], d_key[i+1]);
                PC_2(c_key[i+1], d_key[i+1], K[i+1]);
                { -------------------- }
                l_msg[i+1] := r_msg[i];
                FunctionF(r_msg[i], K[i+1], temp_32);
                XOR32(l_msg[i], temp_32, r_msg[i+1]);
            end;
        InverseIP(l_msg[16], r_msg[16], cryptogram);
    end;
end; { EnDES }
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure DeDES(cryptogram_in : byte_msg_array; key : byte_msg_array;
                var msg_out : byte_msg_array);
begin

end; { DeDES }
{ ------------------------------------------------------------------------ }

begin { Unit Initialisation }

end.
```

```
{ ------------------------------------------------------------------------ }
{   DES simulation unit  ... Kobus Botha  January - May 1990                }
{ ------------------------------------------------------------------------ }

program MagnCryp;
uses Dos, Crt, HouseLib, Win, Win2, DES;

type
    base_type           = (HEX, DECIMAL, BINARY);
    st2                 = string[2];
    mag_ptr             = ^byte;
var
    base                : base_type;
    test_count          : longint;
    medium_key          : array[0..19] of boolean;
    read_write_field    : mag_64;
    crypto_key          : mag_64;
    card_value          : word;
    card_type_code      : word;

const
  c_exit    = ^[;
  c_random  = ^R;
  c_hex     = ^H;
  c_binary  = ^B;

  CClose  = ^C;
  CRight  = ^D;
  CUp     = ^E;
  CEnter  = ^M;
  CInsLin = ^N;
  COpen   = ^O;
  CLeft   = ^S;
  CDown   = ^X;
  CDelLin = ^Y;
```

```
{ ------------------------------------------------------------------------ }
function ReadChar: char;
var
  ch: char;
begin
    ch := ReadKey;
    if ch = #0 then
        case ReadKey of
            #45: ch := c_exit;      { Alt-X }
            #19: ch := c_random;    { Alt-R }
            #35: ch := c_hex;       { Alt-H }
            #48: ch := c_binary;    { Alt-B }

            #24: Ch := COpen;      { Alt-O }
            #46: Ch := CClose;     { Alt-C }
            #72: Ch := CUp;        { Up }
            #75: Ch := CLeft;      { Left }
            #77: Ch := CRight;     { Right }
            #80: Ch := CDown;      { Down }
            #82: Ch := CInsLin;    { Ins }
            #83: Ch := CDelLin;    { Del }
        end;
    ReadChar := ch;
end; { ReadChar }
{ ------------------------------------------------------------------------ }


{ ------------------------------------------------------------------------ }
Function HexStr(Number: byte): st2;
  { The function HexStr converts a byte into a two-character
    hexadecimal number(string) with leading zeroes.   }

    Function HexChar(Number: Word): Char;
    Begin
      If Number < 10 then
        HexChar := Char(Number + 48)
      else
        HexChar := Char(Number + 55);
    end; { Function HexChar }

  Var
    s : st2;
  Begin
    s := '';
    s := s + HexChar(Number div 16);
    Number := Number mod 16;
    s := s + HexChar(Number);
    HexStr := s;
  end; { Function HexString }
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------- }
procedure EnTransp(msg_in : byte_msg_array; key: byte_msg_array;
                           var cryptogram_out : byte_msg_array);
var
    i : integer;
begin
    for i := 0 to 7 do
        cryptogram_out[i] := (msg_in[i] + key[i]) mod 256;
end; { EnTransp }
{ ------------------------------------------------------------------- }


{ ------------------------------------------------------------------- }
procedure EnVigenere(msg_in : byte_msg_array; key: byte_msg_array;
                         var cryptogram_out : byte_msg_array);
var
    i : integer;
begin
    for i := 0 to 7 do
        cryptogram_out[i] := (msg_in[i] + key[i]) mod 256;
end; { EnVigenere }
{ ------------------------------------------------------------------- }


{ ------------------------------------------------------------------- }
procedure DeVigenere(cryptogram_in : byte_msg_array; key: byte_msg_array;
                       var msg_out : byte_msg_array);
var
    i : integer;
    dum : word;
begin
    for i := 0 to 7 do
        msg_out[i] := byte((cryptogram_in[i] - key[i]) mod 256);
end; { DeVigenere }
{ ------------------------------------------------------------------- }


{ ------------------------------------------------------------------- }
procedure ColWrite(s : string; attr : byte);
var
    temp_attr       : byte;
begin
    temp_attr := TextAttr;
    TextAttr := attr;
    write(s);
    TextAttr := temp_attr;
end; { ColWrite }
{ ------------------------------------------------------------------- }
```

```
{ ------------------------------------------------------------------ }
procedure ShowMsg32(in_msg : msg_32; base_ : base_type);
var
    i               : integer;
    temp_str        : string;
    temp_64         : msg_64;
    work_byte_msg   : byte_msg_array;
begin
    case base_ of
        HEX :
            begin
                for i := 0 to 31 do
                    temp_64[i] := in_msg[i];
                MsgBitsToBytes(temp_64, work_byte_msg);
                for i := 0 to 3 do
                    write(HexStr(work_byte_msg[i]), ' ');
            end;
        DECIMAL :
            begin
            end;
        BINARY :
            begin
                temp_str := '';
                for i := 0 to 31 do
                    begin
                        if in_msg[i] = 0 then
                            temp_str := temp_str + '0'
                        else
                            temp_str := temp_str + '1';
                        if (i mod 8) = 7 then
                            temp_str := temp_str + ' ';
                    end;
                Write(temp_str);
            end;
    end; { case }
end; { ShowMsg32 }
{ ------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure ShowMsg64(in_msg : msg_64; base_ : base_type);
var
    i               : integer;
    work_byte_msg   : byte_msg_array;
    temp_str        : string;
begin
    case base_ of
        HEX :
            begin
                MsgBitsToBytes(in_msg, work_byte_msg);
                for i := 0 to 7 do
                    write(HexStr(work_byte_msg[i]), ' ');
            end;
        DECIMAL :
            begin
            end;
        BINARY :
            begin
                temp_str := '';
                for i := 0 to 63 do
                    begin
                        if in_msg[i] = 0 then
                            temp_str := temp_str + '0'
                        else
                            temp_str := temp_str + '1';
                        if (i mod 8) = 7 then
                            temp_str := temp_str + ' ';
                    end;
                Write(temp_str);
            end;
    end; { case }
end; { ShowMsg64 }
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure ShowDES(block : DES_rec; base_ : base_type);
var
    i              : integer;
    screen_line    : integer;
    temp_str       : string;
begin
    with block do
    begin
        GotoXY(1, 1); ColWrite('Msg:  ', $1E);
        ShowMsg64(msg, base_);
        GotoXY(1, 2); ColWrite('Key:  ', $1E);
        ShowMsg64(key, base_);
        screen_line := 3;
        for i := 0 to 16 do
            begin
                Str(i:2, temp_str);
                GotoXY(1, screen_line); ColWrite('LR'+temp_str+' ', $1B);
                ShowMsg32(l_msg[i], base_);
                ColWrite('3', $1F);
                ShowMsg32(r_msg[i], base_);
                inc(screen_line);
            end;
        GotoXY(1, screen_line); ColWrite('Crp:  ', $1E);
        ShowMsg64(cryptogram, base_);
    end;
end; { ShowDES }
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure ShowDESKey(block : DES_rec; base_ : base_type);
var
    i             : integer;
    screen_line   : integer;
    temp_str      : string;
begin
    with block do
    begin
        GotoXY(1, 1); ColWrite('Msg:  ', $1E);
        ShowMsg64(msg, base_);
        GotoXY(1, 2); ColWrite('Key:  ', $1E);
        ShowMsg64(key, base_);
        screen_line := 3;
        for i := 0 to 16 do
            begin
                Str(i:2, temp_str);
                GotoXY(1, screen_line); ColWrite('K '+temp_str+' ', $1B);
                ShowMsg32(l_msg[i], base_);
                ColWrite('3', $1F);
                ShowMsg32(r_msg[i], base_);
                inc(screen_line);
            end;
        GotoXY(1, screen_line); ColWrite('Crp:  ', $1E);
        ShowMsg64(cryptogram, base_);
    end;
end; { ShowDESKey }
{ ------------------------------------------------------------------------ }


{ ------------------------------------------------------------------------ }
procedure BeepNotOk;
begin
    Sound(3000);
    Delay(10);
    NoSound;
    Sound(2000);
    Delay(30);
    NoSound;
end; { BeepNotOk }
{ ------------------------------------------------------------------------ }

{ ------------------------------------------------------------------------ }
procedure BeepOk;
begin
    Sound(6000);
    Delay(1);
    NoSound;
    Sound(12000);
    Delay(1);
    NoSound;
end; { BeepOk }
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
procedure RandomEnDES(var des : DES_rec);
var
    i   : byte;
begin
    repeat
        inc(test_count);
        GotoXY(64, 21); write(test_count:10);
        with des do
            for i := 0 to 63 do
                begin
                    msg[i] := Random(2);
                    key[i] := Random(2);
                end;
            EnDES(des);

            { .. Display --------. }
            ShowDES(des, base);
    until KeyPressed;
end; { RandomEnDES }
{ ------------------------------------------------------------------------ }

{ ------------------------------------------------------------------------ }
procedure Initialize;
begin
  CheckBreak := False;
  if (LastMode <> CO80) and (LastMode <> BW80) and
    (LastMode <> Mono) then TextMode(CO80);
  TextAttr := $3F;
  Window(1, 2, 80, 24);
  FillWin($177, $72);
  Window(1, 1, 80, 25);
  GotoXY(1, 1);
  Write(' Magnetic card  -  Crypto Simulation ');
  ClrEol;
  GotoXY(1, 25);
  Write(' Alt-H-Hex   Alt-R-Random ' +
    ' Alt-I-Exit  Alt-R-Random  Esc-Exit ');
  ClrEol;
  Randomize;
  TopWindow := nil;
  WindowCount := 0;
end;
{ ------------------------------------------------------------------------ }
```

```
{ ------------------------------------------------------------------------ }
{ Main Program                                                             }
{ ------------------------------------------------------------------------ }
var
    clear_text_window   : WinState;
    vigenere_window     : WinState;
    i, j                : integer;
    screen_line         : integer;
    ch                  : char;
    dum_str             : string;
    dum                 : byte;
    temp_str            : string;

    des_1               : DES_rec;
    temp_32             : msg_32;
    done                : boolean;
begin { Main }
    { ------------------- }
    Randomize;
    test_count := 0;
    ClrScr;
    Initialize;
    OpenWindow(1, 2, 79, 24, '   DES    ', $1F, $71);
    FillWin(chr($DB), $01);
    SaveWin(clear_text_window);

    GotoXY(1,1);
    (* write('         Message         3          Key          3   Vigenere Cryptogram '); *)
    ClrEol;
    WriteStr(50, 21, ' Test Count : ', $17);
```

```
    { ------------------- }
    done := False;
    base := BINARY;
    repeat
        begin
            ch := ReadChar;
            BeepOk;
            case ch of
                c_exit: done := true;
                c_random: RandomEnDes(des_1);
                c_hex:
                    begin
                        base := HEX;
                        CloseWindow;
                        Window(1, 1, 80, 25);
                        GotoXY(1, 25);
                        Write(' Alt-B-Binary   Alt-R-Random ' +
                            ' Alt-X-Exit  Alt-R-Random  Esc-Exit ');
                        OpenWindow(1, 2, 79, 24, '   DES    ', $1F, $71);
                        ClrScr;
                        ShowDES(des_1, base);
                    end;
                c_binary:
                    begin
                        base := BINARY;
                        CloseWindow;
                        Window(1, 1, 80, 25);
                        GotoXY(1, 25);
                        Write(' Alt-B-Hex      Alt-R-Random ' +
                            ' Alt-X-Exit  Alt-R-Random  Esc-Exit ');
                        OpenWindow(1, 2, 79, 24, '   DES    ', $1F, $71);
                        ClrScr;
                        ShowDES(des_1, base);
                    end;
            else
                BeepNot_k;
            end;
        end;
    until done;
    Window(1, 1, 80, 25);
    NormVideo;
    ClrScr;
end.
{ ================================================================================ }
```

Appendix B

Companies and Products

Product	Company
high-coercivity magnetic card	*GEC Traffic Automation* *Brown, Davis & McCorquodale*
Magnetic Watermark tape	*Thorn EMI Electronics*
Autelcard card reader	*Autelca*
optical card reader	*Sodeco*
chip-memory cards	*Flonic* *Epson* *Fujisoku* *Dallas Semiconductor* *Du Pont Connector Systems* *Mitsubishi Electronics America Inc.*
fused link card	*Tadiran*
contacted smart card	*Allied Vector Systems* *Smart Card International Inc.* *Honeywell Bull* *Flonic-Schlumberger* *Philips SA* *Catalyst Semiconductor, Inc.*
contactless smart card *Indicard*	*GEC Card Technology* *Tadiran* *Industrial Software Engineering CC, Pretoria*
supersmart card	*Toshiba* *Visa*
magnetic tape	*Kurz*
magnetic head	*Magnetic Components*
card telephone	*Telkor* *Palindent Ltd.*

References

[1] Weil, Andre. *Number Theory.* Birkhauser, 1983. (ISBN 3-7643-3141-0, ISBN 0-8176-3141-0)

[2] Van Tilborg, Henk C. A. *An Introduction to Cryptology.* Kluwer Academic Publishers, 1987. (ISBN 0-89838-271-8)

[3] Miles E. Smid and Dennis K. Branstad. *The Data Encryption Standard : Past and Future.* Proceedings of the IEEE, Vol. 76, No. 5, May 1988. (ISBN 7198 2611 X Hardback, 7198 25717 Paperback.)

[4] Henry Beker and Fred Piper. *Cipher Systems.* Northwood Books, London, 1982.

[5] Dewdney, A. K. *Computer Recreations.* Scientific American, November 1988.

[6] James L. Massey *An Introduction to Contemporary Cryptography.* Proceedings of the IEEE, Vol. 76, No. 5, May 1988.

[7] Claassen, G. J. *Overview of Information Security Standards.* Infosec '89 Proceedings, Microelectronics and Communications Technology, CSIR.

[8] Laurie, D. P. *'n Inleiding tot Openbare Sleutelkriptostelsels* Infosec '89 Proceedings, Microelectronics and Communications Technology, CSIR.

[9] Von Solms, S. H. *Syferhandtekeninge* Infosec '89 Proceedings, Microelectronics and Communications Technology, CSIR.

[10] Sloane, N. J. A. *Error Correcting Codes and Cryptography* The Mathematical Gardner

[11] Simmons, Gustavus J. *A Survey of Information Authentication* Proceedings of the IEEE, Vol. 76, No. 5, May 1988.

[12] Diffie, Whitfield *The First Ten Years of Public Key Cryptography* Proceedings of the IEEE, Vol. 76, No. 5, May 1988.

[13] Harrop, Peter *New Electronics for payment* IEE Review, October 1989.

[14] *Telecommunication Products* Palindent Ltd. (Unpublished).

[15] Myers, W. *Micro View, The Road to the Supersmart Card* IEEE Micro August 1989.

[16] Weinstein, Stephen B. *Smart Credit Cards : the answer to cashless shopping* IEEE Spectrum, February 1984.

[17] *Intelligent Data Carriers - An Introduction to the Smart Card* Motorola.

[18] Terry, Chris *Technology Update : Smart cards are getting smarter and faster* EDN, March 16, 1989.

[19] Simmons, Gustavus J. *Scanning the Issue* Proceedings of the IEEE, Vol. 76, No. 5, May 1988.

[20] Van Wyk, J. D. N. *Cryptology – The Scenario* Infosec '89 Proceedings, Microelectronics and Communications Technology, CSIR.

[21] Kühn, G. J. *A Review of Cryptographic Algorithms* Infosec '89 Proceedings, Microelectronics and Communications Technology, CSIR.

[22] Security Report on GEC Parking meter — Castle Rock Consultants Ltd. (Unpublished)

[23] *GEC Contactless Smart Card makes a Technological Breakthrough* GEC Card Technology press release.

[24] Barret, Paul *Implementing the Rivest Shamir and Adleman Public Key Encryption Alogithm on a Standard Digital Signal Processor* Advances in Cryptology – CRYPTO '86 Proceedings, Ed A. M. Odlyzko, Springer Verlag, 1986.

[25] Desmedt, Yvo, and Quisquater, Jean-Jacques *Public-Key Systems based on the Difficulty of Tampering (Is there a difference between DES and RSA?)* Advances in Cryptology – CRYPTO '86 Proceedings, Ed A. M. Odlyzko, Springer Verlag, 1986.

[26] *Tadicard — The contactless smart card for many applications* Sales brochure from Industrial Software Engineering CC, Pretoria.

[27] Brickell, Ernest F., and Odlyzko, Andrew M *Cryptanalysis: A Survey of Recent Results* Proceedings of the IEEE, Vol. 76, No. 5, May 1988.

[28] GEC Card Technology Information Sheet, Ref: PD/05/1-1

[29] Knuth, Donald E. *The Art of Computer Programming, Volume 2 – Seminumerical Algorithms* Addison-Wesley 1969.

[30] *ANSI Standard for financial services - financial transaction cards - magnetic stripe encoding* ANSI X4.16-1983

[31] Jorgensen, Finn *The Complete Handbook of Magnetic Recording* TAB Books, 1980. ISBN 0-8306-9940-6

[32] Magnetic Components (Chertsey, England) Partial Specification for magnetic High-coercivity Card Head – Drawing number B-0L-17376

[33] *The Heritage Illustrated Dictionary of the English Language* American Heritage Publishing Co., Inc., 1975. ISBN 07-001173-7

[34] Zimmermann, Philip *A proposed Standard Format for RSA Cryptosystems* IEEE Computer, September 1986.

[35] Brassard, Gilles, and Crepeau, Claude *Non-transitive Transfer of Confidence: A Perfect Zero-knowledge Interactive Protocol for SAT and Beyond* 27th Annual IEEE symp. on the Foundations of Comp. Sci.; pp 188 – 195, 1986.

[36] Fiat, Amos, and Shamir, Adi *How to prove yourself: Practical Solutions to Identification and Signature Problems* CRYPTO '86 Proceedings, Ed A. M. Odlyzko, Springer Verlag, 1986.

[37] Chaum, D., et al *Demonstrating Possession of a discrete logarithm without revealing it* CRYPTO '86 Proceedings, Ed A. M. Odlyzko, Springer Verlag, 1986, ISBN 3 540-18047-8.

[38] *Functional Specification for a Card Reader System used in Payphone Application* Spec. A4/31116 GEC Traffic Automation Limited, 1987.

[39] Broomberg, B. *Smartcards: How, What, Why and When* Infosec '90 Proceedings, Microelectronics and Communications Technology, CSIR.

[40] ISO Standard 7811 – 7813 *Identification cards – financial Transaction cards*

[41] *Autelcard – Introduction to Watermark Magnetics* Autelca (unpublished)

[42] Kühn, G. J., Bruwer, F. J. and Smit, W. *'n Vinnige veeldoelige Enkripsievlokkie* Infosec '90 Proceedings, Microelectronics and Communications Technology, CSIR.

Index