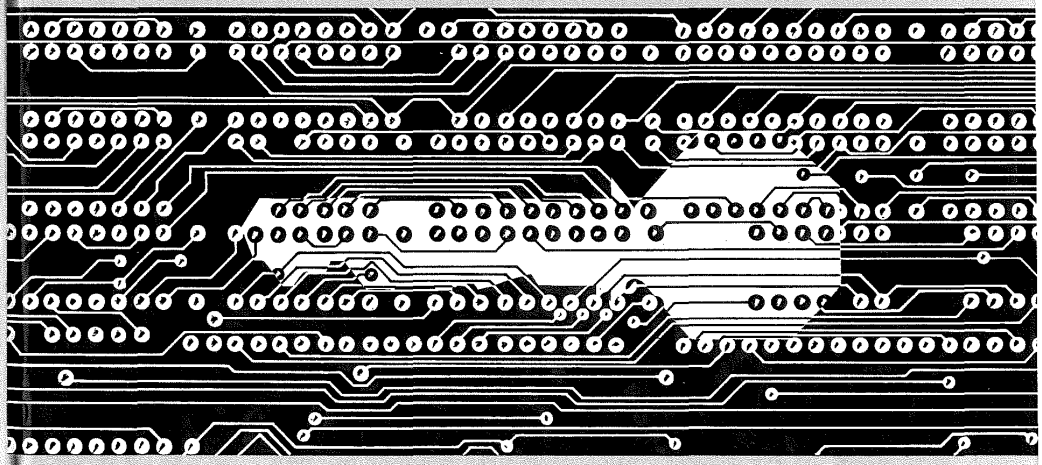
Security for Computer Networks

Second Edition

An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer



D.W. Davies and W.L. Price

SECURITY FOR COMPUTER NETWORKS

WILEY SERIES IN COMMUNICATION AND DISTRIBUTED SYSTEMS

Satellite Communications Systems
G. Maral and M. Bousquet

Integrated Digital Communications Networks (Volumes 1 and 2)
G. Pujolle, D. Seret, D. Dromard and E. Horlait

Security for Computer Networks, Second Edition
D.W. Davies and W.L. Price

SECURITY FOR COMPUTER NETWORKS

An Introduction to Data Security in Teleprocessing and Electronic Funds Transfer

Second Edition

D.W. Davies
Data Security Consultant

and

W.L. Price
National Physical Laboratory, Teddington, Middlesex

JOHN WILEY & SONS
Chichester · New York · Brisbane · Toronto · Singapore

Wiley Editorial Offices

John Wiley & Sons Ltd, Baffins Lane, Chichester,
West Sussex PO19 1UD, England

John Wiley & Sons, Inc., 605 Third Avenue,
New York, NY 10158-0012, USA

Jacaranda Wiley Ltd, G.P.O. Box 859, Brisbane,
Queensland 4001, Australia

John Wiley & Sons (Canada) Ltd, 22 Worcester Road,
Rexdale, Ontario M9W 1L1, Canada

John Wiley & Sons (SEA) Pte Ltd, 37 Jalan Pemimpin #05-04,
Block B, Union Industrial Building, Singapore 2057

Library of Congress Cataloging-in-Publication Data:

Davies, Donald Watts.
Security for computer networks : an introduction to data security in teleprocessing and electronic funds transfer / D.W. Davies and W.L. Price. — 2nd ed.
p. cm. — (Wiley series in communication and distributed systems)
Includes bibliographies and index.
ISBN 0 471 92137 8
1. Data transmission systems—Security measures. I. Price, W. L. II. Title. III. Series.
TK5105.D43 1989
005.8—dc20 89-14760
CIP

British Library Cataloguing in Publication Data:

Davies, D. W. (Donald Watts, *1924*—)
Security for computer networks : an introduction to data security in teleprocessing and electronic funds transfer. — 2nd ed — (Wiley series in communications and distributed systems)
1. Computer systems. Security measures. Cryptograms
I. Title II. Price, W. L.
005.8'2

ISBN 0 471 92137 8

Typeset by MHL Typesetting Ltd, Coventry
Printed and bound in Great Britain by Courier International, Tiptree, Essex

To Diane and Marjorie

CONTENTS

ACRONYM LIST

ABA	American Bankers Association
ANSI	American National Standards Institute
AP	authentication parameter
ASCII	American Standard Code for Information Interchange
ATM	automatic teller machine
BACS	Bankers' Automated Clearing Services
BSC	binary synchronous communication
CBC	cipher block chaining
CCITT	International Telegraph and Telephone Consultative Committee
CFB	cipher feedback
CHAPS	Clearing Houses Automated Payments System
CHIPS	Clearing Houses Inter-bank Payments System
CLCB	Committee of London Clearing Bankers
CRC	cyclic redundancy check
DCE	data circuit terminating equipment
DEA	Data Encryption Algorithm
DEE	data encryption equipment
DES	Data Encryption Standard
DK	data key
DMA	direct memory access
DSA	decimal shift and add
DTE	data terminal equipment
EAROM	electrically alterable read-only memory
ECB	electronic code book
EDC	error detection code
EFT	electronic funds transfer
FAR	false alarm rate
FIPS	Federal Information Processing Standard
HDLC	high level data link control
IPR	impostor pass rate
ISO	International Organization for Standardization
ITAR	International Traffic in Arms Regulations
IV	initializing variable
KGM	key generation module
KK	key encrypting key
KKM	master (key encrypting) key
KLM	key loader module
KTM	key transport module

lcm	least common multiple
LSI	large scale integration
MAC	message authentication code
MAR	MAC residue
MDC	manipulation/modification detection code
NBS	National Bureau of Standards
NIST	National Institute of Standards and Technology
NPL	National Physical Laboratory
NSA	National Security Agency
NSM	network security module
OFB	output feedback
OSI	Open Systems Interconnection
OWF	one-way function
PCM	pulse code modulation
PIN	personal identification number
PKC	public key cryptosystem
p-MOS	p-channel metal oxide semiconductor
PROM	programmable read-only memory
PSS	Packet Switchstream
PVV	PIN verification value
RAM	random access memory
ROM	read-only memory
RSA	Rivest, Shamir and Adleman (PKC)
SDLC	synchronous data link control
SID	S.W.I.F.T. interface device
SNA	systems network architecture
S.W.I.F.T.	Society for World-wide Inter-bank Financial Telecommunications
TRM	tamper resistant module
TTL	transistor-transistor logic
XOR	exclusive-OR

PREFACE TO THE FIRST EDITION

Blackmail, fraud and the stealing of commercial secrets are examples of crimes using information as the medium. There was a time, not many years ago, when it was necessary to alter the account books in order to cover up a fraud; now the same effect can sometimes be produced at a computer terminal. The widespread introduction of information technology into business inevitably leads to its misuse for crime, which data security aims to prevent.

Essentially, security means controlling access to data, depriving the blackmailer of his information, protecting commercial secrets and preventing the falsifying of records. The need to control access in computers first became serious when 'time-sharing' began to operate. Undergraduates regarded the security features of these systems as a challenge and soon found that the existing methods of control were defective. This taught designers a useful lesson: that data security needs care and attention.

In 1983 the film *War Games*, and the publicity it created, introduced people to a new and surprising cult, the 'computer hackers' who spend their time obsessively trying to break into computer systems. Their devotion to a basically tedious pastime is extraordinary, and it shows up one advantage of an amateur attacker over the professional defenders. The attacker's time is apparently unlimited and uncosted; the defender's time is expensive. The systems that hackers have broken into were protected only by weak password schemes, but the hackers' success and the excitement it generated prepares the way for more advanced attacks when simple hacking fails to satisfy them. A special feature of illegal attacks on computer systems is that there is no tradition of associated guilt-feelings. The law, in most countries, has not begun to define the new kinds of crime. With no likelihood of punishment, the probing of the defences of computer systems is considered to be a game. It could become the tool of organized crime.

Cryptography is a long-established way to keep information secret. Now that the majority of information systems transport data from place to place (as well as processing and storing it) the place of cryptography in data security is assured. It is beginning to be accepted that cryptography is the basic tool for achieving data security. This book is about the use of cryptography to protect data in teleprocessing systems, not only keeping data secret but also *authenticating* it, preventing alteration and proving its origin. These go beyond the classical uses of cryptography. Some of the main concerns of data security in banking and other commercial information systms are those of authentication.

After the introduction in Chapter 1, ciphers are described in Chapter 2 and then the Data Encryption Standard in Chapter 3. This standard is used as one of the principal tools of data security in what follows. The first step is to define a number

of standard ways of using the cipher — extending its properties and making it more useful. These are the 'modes of operation' described in Chapter 4. The application of ciphers to the authentication of messages is the subject of Chapter 5. Each application needs a scheme to manage encipherment keys and two of these key management schemes are described in Chapter 6.

Teleprocessing systems are used by people at widely spread terminals. Only if the identity of these people can be determined at a distance will it be possible to control access to data. Verification of personal identity is the subject of Chapter 7.

A breakthrough in encipherment technique occurred in 1976 with the discovery of the 'public key ciphers' by Diffie and Hellman. The properties of these ciphers make them specially useful for access control and they have unique virtues for key distribution and for a strong form of authentication called a 'digital signature'. Chapter 8 describes public key ciphers and Chapter 9 describes digital signatures.

The largest commercial use of cryptography and other data security techniques is in banking and the payment systems operated by banks. Chapter 10 describes how data security measures can be applied to 'electronic funds transfer' in its various forms, leading us to some new concepts for future payment systems.

Finally, the wide use of these methods will only be possible if systems can interwork and a range of compatible equipment can be manufactured. Chapter 11 describes the progress made towards standards for data security.

We have tried to make the approach pragmatic, illustrating the principles with examples. At present the subject is essentially pragmatic in that only the insecurity of systems can be demonstrated, by breaking them, but their security is demonstrated only by the absence of demonstrated flaws.

The information in this book represents the best practice known to us but it cannot carry any guarantee of security — nothing can. When a cipher, or another part of a security scheme, resists attacks for a long time, this gives the users confidence, but there is no absolute or provable basis for that confidence. The possibility of provable security (in some sense yet to be defined) still exists but, with a few exceptions, proofs elude us. In this respect there has been no big change as a result of new technology — the security of ciphers has always been tenuous and this is still true of nearly all aspects of data security.

Mathematical notation is often the clearest way to describe unambiguously how a cipher operates, or how it is used. Some ciphers, notably public key ciphers, cannot be described in any other way. Whenever possible in this book, intuitive arguments have been given as well. We aim in this book to help those people who will actually put security schemes into practice. More formal and complete treatments will sometimes be found in the original papers listed in references at the end of each chapter.

these may continue to elude us) there are now a few schemes with well-developed security that make them worthwhile examples of how this difficult environment can be dealt with. We have added descriptions of three schemes, the transaction key, derived unique key per transaction and the use of public key cryptography.

We have taken the opportunity to bring some of the terminology in line with international standards, but without losing touch with accepted terminology. Also, a set of protocols for authentication, associated with the names of Fiat and Shamir, are described in this new edition.

Because this book is mainly concerned with the application of new technology to information security and deals with basic methods rather than specific implementations, the ideas it describes are remarkably stable.

The second edition has allowed us to make worthwhile changes and additions to the text while the main shape of the book and the concepts it introduces have remained unchanged.

Finally, we have received no solutions for the cryptogram given at the end of the preface to the first edition and reproduced here.

ZSYTT	IFXVJ	NVWRS	OHTTR	ESUIJ	VGEJB	DDO.A	TZMKV
QUYVS	QTMVX	NGDVX	.AWIP	TKVTZ	KMMSG	DVITF	NKXRF
BDYDL	HHRDK	DNTQE	AKJDK	HGFST	DU.MO	CCNXY	

with well-developed
ifficult environment
mes, the transaction
c key cryptography.
inology in line with
d terminology. Also,
s of Fiat and Shamir,

n of new technology
rather than specific
ible.
anges and additions
ts it introduces have

n given at the end of

DDO.A TZMKV
DVITF NKXRF
CCNXY

Chapter 1 DATA SECURITY

1.1. THE NEED FOR DATA SECURITY

Each major advance in information technology changes our ideas about data security. Consider the use of written mesages to replace those carried in the memory of a messenger. Written messages are less prone to error, and since the courier need not know the message, but can destroy it in an emergency, there could be better security. Yet written messages can be concrete evidence of conspiracy or spying, whereas a carrier of a spoken message might escape unsuspected. The need to hide the content of a written message must therefore have been realized very soon, and there is evidence that codes and ciphers appeared almost with the beginning of writing.

When telegraphy began, Wheatstone and others thought that ciphers would be needed by the senders of telegrams and there was a flurry of new ideas for cryptography. But the need did not show itself. Cryptography has traditionally been used by kings, popes, armies, lovers and diarists and has moved into business and commerce only recently. Now that information is processed, stored and transmitted in large quantity, the need for data security is greater and more varied. We shall be dealing with the secrecy of transmitted information, using encipherment, and also with authentication of information, verifying the identity of people, preventing the stealing of stored information and controlling access to both data and software. Such varied data security problems arise from the more recent advances in information technology — large stores and microprocessors which give us processing wherever we need it. The microprocessor and the floppy disc create a new and urgent problem of safeguarding software. The requirements summed up by the phrase 'data security' do not stay the same; they change as the technology changes.

Nowadays, almost everything we do with information is assisted by some new invention. The fundamental operations of storage, processing and transmission seem to be undergoing a relentless and rapid improvement, so fast that the application of the technology cannot keep up with the rate of advance. The introduction of teleprocessing into all aspects of business, commerce and industry brings new data security problems to the fore. Undoubtedly, electronic banking has experienced the first impact, with spectacular frauds in the shape of falsified money transfers. Banks have been the first to apply modern methods to make their systems secure. The rest of commerce and industry is well behind banking in its attention to data security.

In order to avoid repeating the same phrases we will use conventional names for the actors in the security drama. The bad guys who are trying to do something

with the system, which the designers would like to avoid will be called 'the enemy' and their activities will be called 'attacking' the system.

These words have only a conventional meaning. The enemy may not, for example, be a remorseless or professional adversary. For a long time, students were the principal finders of weaknesses in operating systems, because they enjoyed the intellectual challenge. Almost certainly, a lot of the effort that went into 'phone freaking' was motivated in this way. None the less, it is potentially dangerous because the techniques developed in innocent attacks can be exploited by criminals.

New methods of payment and transfer of funds using communication networks and intelligent tokens are developing fast and are usually called 'electronic funds transfer'. These are an obvious target for fraudulent manipulation of data. As electronic message systems of all kinds come into wider use, carrying data which has already been captured in the 'electronic office', there must inevitably be a proportion of sensitive information carried as 'electronic mail'. This is another area which needs attention to security, both for secrecy of information and to preserve its authenticity.

The advance of information technology has sparked off a public debate on the subject of individual privacy. Like many public debates it expresses both rational and irrational fears. Everyone is now alerted to the real dangers of inaccurate personal information and uncontrolled access to files. Most advanced countries have introduced laws to enforce a reasonable degree of individual data privacy and others have such laws in preparation. If we think of *privacy* as the legal concept, then *security* of data is one of the means by which the privacy can be obtained. The implementation of these new laws is likely to produce applications for data security techniques.

Of the three main operations carried out in information systems, storage, processing and transmission, it is undoubtedly data transmission that carries the greatest of security risks. A communication network consists of numbers of cables, radio links, switches and multiplexers in a variety of locations, all of these parts of the system being potential targets for 'line taps' or 'bugs'. It is impossible to make a widespread network physically secure, therefore security measures depend on information processing techniques such as cryptography.

Storage of data is next on the list of vulnerabilities because data spends much more time in storage than in processing. The protection of stored data can use the same techniques of cryptography but with different twists. For most purposes, processing is the least vulnerable part but it may be worth attacking. Perhaps an enemy can gain more by discovering which parts of a file are active than by stealing a whole file. For example a police file of intelligence about crime activities contains a lot of data about dormant criminals and perhaps even more about dubious suspects. A team contemplating a big crime could learn more from the data being accessed than by searching the whole data base.

No data system can be made secure without physical protection of some sort of the equipment. The effect of good design is to concentrate the need for physical security, not to circumvent it entirely. In particular, processing of data usually (perhaps always) requires those data to be in clear form, not enciphered, therefore processors themselves must be protected from intrusion, such as the attachment of radio bugs. In many systems the data and operations needing the greater degree

of security can be contained in one box of modest size, physically strong and designed to destroy its stored secrets when it is opened. This is called a *tamper-resistant module* and it is a central element of many systems which this book will describe.

A tamper-resistant module typically protects a process which is essential to the security of the system such as the handling of cryptographic keys and the cryptographic operations which they enable. Usually there is a procedure to be performed inside the module and this requires a small processor. The interface between the module and the rest of the system must be designed carefully to avoid any compromise of secret data and this interface is also controlled by the processor within. The software employed within the module must be trusted — software integrity is discussed in the next section. The module is protected by being enclosed in a strong box but usually the strength of this box cannot ensure that it will never be broken into. Consequently, the module is provided with a number of sensing devices which will detect various forms of tampering. These are sensitive to vibration, penetration of the protective box, tilting or temperature, for example. When the sensors detect an attack, at a certain level of severity all the important data contained within the enclosure are destroyed by over-writing. By this technique of 'tamper-responding' the secret data can be protected without an exceptionally strong outer case. Details of the techniques for making tamper-responding modules are best kept confidential because knowledge of the sensing devices helps an attacker to overcome them and this increases the cost for future generations of tamper-resisting modules.

1.2. ASSESSMENT OF SECURITY

Security is a complex property and difficult to design or optimize. Designing a system to be efficient, convenient or cheap is an optimization problem which, though complex in practice, has a mathematical structure which is easy to understand. The problem is to choose the design parameters in order to maximize a figure of merit. Designing a system for *security* means analysing an adversary problem where the designer and the opponents are each independently thinking out their strategies. The outcome of the contest is a result of their combined choices. The mathematical theory for such problems is the 'theory of games'. The kind of game which fits the situation is a 'two-person, non-zero-sum game with imperfect information'. The 'non-zero-sum' condition appears because the system attacked may lose more than the attacker gains. 'Two-person' means that we consider one attacker at a time, to simplify the problems as much as possible. Games theory, beyond the most trivial cases, is extremely difficult and there seems to be no way in which the theory of games can actually be useful in analysing security — it merely serves to illustrate the underlying complexity of the problem and the inadequacy of a naive 'risk analysis' approach.

Every kind of threat to a system should be assessed, yet it is very difficult to enumerate the complete range of attacks. First of all, the motives and intentions of the attackers have to be guessed. Stolen information can be used for fraud, espionage, commercial advantage, blackmail, and probably in many other ways. Enciphering information on a magnetic tape can make the tape useless to an enemy as information, but, if there are no other copies of this tape and the information

is essential to its owner, the stolen tape can be used as a hostage, to extort payment for its return. This was a threat used in a crime in Europe, where an ex-employee stole the tapes from a large company's computer centre. A great deal of imagination is required to think out all the ways in which a system could be attacked. In a bank in the USA, each cheque book was supplied with magnetically encoded forms for paying money into its account. If the customer had forgotten to bring his forms, he could pick up similar forms in the banking hall, which were not encoded. At the first stage of sorting according to the magnetic characters, uncoded forms were rejected, then operators inserted the account number of the payee along with the other details that had been written on the form. An ingenious fraud was devised by distributing coded paying-in forms among the forms in the banking hall. Customers did not notice the presence of the coding and unwittingly paid their money into the fraudster's account.

The existence of so many methods of attack makes the protection of an information system very difficult indeed. It is doubtful that a systematic method to analyse the problem will ever be found, though checklists can be a help. It is unusual to have an information system described in sufficient detail that all the potential weaknesses can be identified. Therefore, a good investigator works from observation and by discussion with those who know the system well, not from paper designs and specifications. The ability to find security flaws depends on an attitude of mind which takes nothing on trust.

Security is essentially a negative attribute. We judge a system to be secure if we have not been able to devise a method of misusing it which gives some advantage to the attacker. But we might just have failed to identify the nature of the enemy or missed out a method of attack.

A security investigation should never be based on the designer's concept of the system. He has thought about the security and convinced himself that every aspect has been covered. Probably, he has a good theoretical reason for believing that the security goals have been achieved. What is then required is 'lateral thinking' which questions his assumptions and finds different approaches. The designers may have concentrated their attention on some parts of the system and forgotten others. On the other hand, systems can be so complex that only those who implemented them can fully understand their details. With a complete change of attitude, the implementers can be valuable allies in a security study. They must first be convinced that there are flaws and persuaded to search for them.

The security measures applied to a system always cost something and in an extensive network the total expense may be very large. Before embarking on a programme of installing security features, managements will want to know that the expense is justified. For this purpose elaborate procedures have been developed, called *risk assessment*. A number of different threats are tabulated, together with measures of the probability that each will happen and the likely losses that would arise from a successful attack. From these data, in an obvious way, the 'annual loss expectancy' associated with each type of threat can be evaluated. The idea behind this assessment is that a comparison of the loss expectancy with the cost of providing protection will be a guide for the managers in deciding what security features to install.

The data which enter into a risk assessment are known only with very low accuracy, rarely better than an order of magnitude and sometimes even less preci-

sion. Consequently, risk analysis can hardly be considered a scientifically based procedure. But managements need some guidance and the many risk-assessment methodologies on the market offer them a degree of help.

There cannot be any prescription for obtaining data security. The system designer could miss one vital point, perhaps one which had nothing to do with information technology. For example, an automatic teller machine (no longer in production) delivered bank notes through a slot which was contained in a neat recess. Someone built an attractive looking plate which covered the recess and presented to the user a dummy slot from which the money would never emerge. He fitted this plate to an automatic teller outside a bank, when the bank was open, stood back and watched what happened. The user went through the correct procedure but received no money, then went into the bank to complain. The crook took away his plate and the money which was concealed behind it. This is a variant of an old trick which has been used with returned-coin chutes since vending machines and public telephones began; it pays off better with automatic teller machines. Some of the earliest attacks on automatic teller machines were to pull them out of the bank wall, illustrating that the security of a system may have nothing to do with its data handling.

When all the precautions have been taken against illegal access to data in communications and storage, there remains an indefinite number of other threats. There are two areas of sufficient importance for special mention, system software and the people operating the system.

Software integrity

The complexity of an information system is built mainly into its software, in order that the hardware can be simpler and composed mainly of standard units such as microprocessors and stores. The virtue of software is that it can be changed both during development and subsequently to give the system new properties. This flexibility is a severe threat to system security.

The first difficulty with software is understanding it sufficiently to be sure that it functions correctly in the first place. Where equipment is built with several processors, each handling a restricted range of functions and interacting with others according to carefully devised rules or protocols, the verification of software may be a tractable problem. At the other extreme, mainframes with their complex operating systems are never fully understood. If the security requirements are strict, it must be assumed that any normal operating system has weaknesses. In particular these flaws could be exploited by the software designers, who know the details well enough to find the flaws. In this respect, large operating systems have improved a little and the provision of special hardware to assist access control has made them proof against attacks which were common in the early days but no complex system can be regarded as entirely secure, even if the designers and builders were all trustworthy and had only that aim in mind.

A dishonest designer could provide 'hooks' in the software ready for the attachment of modifications to undermine its security. The auditing of software by independent software designers should look for unnecessary features which might be used in this way.

During software design, special tools are provided to help the implementers

modify programs at various levels, from binary patches up to the top level of specification. The very convenience of these tools makes them dangerous, particularly if a software writer is working for an enemy. Retaining these tools when the software is in operation constitutes a major security weakness because it allows post-design modifications which could introduce deliberate trapdoors into the security-handling parts of the system. The severity of this threat is greater if it is planned early in the system design process, hence the importance of trustworthy system designers. There may be a need to keep one or two highly instrumented versions of the system for software testing and updating but those systems which go outside a secure environment should be made as difficult to modify as possible.

If the software of a system is entirely reliable and trustworthy, the next problem for the designer is how to keep it that way. Some software can be physically protected, for example by storing it in a read only memory (ROM) or in a store which, though it could be overwritten is protected by a physical 'write protect' device. Preventing the over-writing of software by a programmed feature (such as an access control mechanism in the operating system) is no real protection because that can be altered. The more ingenious attacks on software integrity employ the operating system. This is not only the most powerful part of the software but also the least understood and therefore the most difficult place for the discovery and prevention of attacks.

Dedicated processors, carrying out a single task with a fixed program, can be reasonably secure if they are protected against physical attack.

The least secure program environment today is the microcomputer because the philosophy of designing these systems is to make them highly adaptable, with the greatest ease for changing their programs. Users of microcomputers can obtain free software from a number of sources. If they accept and run it they can endanger their entire system, for example doing irreparable damage to the contents of their fixed disc. This attack by a program which appears innocent and does something useful but contains a hidden enemy is called the *Trojan horse* attack.

Even more insidious is the software *virus*. This is designed to infect other programs so that the treacherous program is replicated and, in some cases, moves from one computer to another.

Just before Christmas 1987, in West Germany, a Christmas greeting message was sent into an electronic mail system on a European Academic Research Network. The message produced a pretty display and invited the user to run the program. When this was done, the program accessed the standard files containing the names of the user's correspondents in the mail system and sent out a copy to all of these people. Though basically a harmless type of virus, it spread through at least three networks and the quantity of electronic mail it produced swamped the networks' capacity.

More insidious viruses can spread in the same way and remain dormant until some criterion is met (such as time and date) when they begin a destructive phase, such as over-writing storage. In order to remain dormant and yet spread from program to program, they can be hidden in the operating system or in parts of files which are employed by the operating system but invisible to the user.

Maintaining the integrity of software is not technically difficult but it can conflict with the way in which computer systems are used and with the freedom to make changes or adopt software from any source. It is not only the software

that needs to be protected. The security properties of a system can also be incorporated in tables giving the rights of access to files for each user. These tables must be protected as carefully as the software itself. The tool for protecting the contents of any file against unauthorized access is authentication, which is the subject of chapter 5. The authentication mechanism itself cannot be in software; therefore an unmodified general purpose system such as the typical personal computer is never a secure environment.

Security and people

The owners of a system depend for its security in the first place on the integrity of the supplier, which itself depends on the people who design, build and maintain the system. When it is in operation, keys and passwords are introduced which protect it from enemies outside, including enemies in the ranks of the suppliers, unless software changes have subverted the protection. With good design, the security then depends on the people who operate the system and carry out its security-related procedures.

Some operators will be in positions of very great trust, such as those who load the system with master keys and those who transport keys from one part of the system to another. Improved design can reduce the need for some of this trust, up to a point. For example, the transport of cryptographic keys can use hardware modules which make the keys inaccessible to the couriers. With these means, it requires a much more elaborate attack to obtain the keys illegally, though it is never impossible. Some kinds of privileged operation cannot be avoided. Someone has to write into the system the rules of access control at the top level, giving special privilege to certain people, such as database managers or security controllers. Careful design can reduce the residual threat from these directions. For example, a security controller may be responsible for enforcing the correct procedure in using the system but he need not know the values of any cryptographic keys. In this way, though some trust is placed in a number of people, there are few opportunities for individuals to subvert the whole system. Where there is a particularly sensitive requirement, such as a top-level master key, the responsibility can be split between several individuals, with the effect that it would require a conspiracy of all of them to undo the security of the system as a whole. In a well-designed system it should be clear who is being trusted and to what extent, but there is no way to make the system proof against unlimited deceit.

1.3. THE EFFECTS OF TECHNOLOGY

As information technology becomes more complex, the opportunities for its misuse are greater. Complexity of design is not a protection against interference with data systems. At first sight it might seem that a complex protocol makes it more difficult to extract information from a communication link or change data in transit. In practice, increased complexity is made workable by greater standardization and then standard software and hardware becomes available to unravel the complexities. For example, it is now possible to buy instrumentation which will interpret the binary pattern on a communication line, enabling a specialist to determine the link layer protocol and extract the contents of frames, also to interpret

the packets of the X.25 protocol and the transport, session and document layers of the Teletex procedure. With this compact equipment, all the complexities of many levels of protocols can be stripped away for diagnostic purposes. Using the same equipment, an enemy can interpret what is passing on the line and, with some additional software, can capture information, alter it and re-launch it with the correct format and procedure. This is not easy but it is not made prohibitively difficult just by protocol complexity.

In fact, the variety of new technologies increases the possibilities of attack. If a local ring or Ethernet is used, all of the data passing between terminals and computing resources in this local area can be made accessible from a single line tap. The Ethernet broadcasts all messages to all terminals. In most designs of ring, all the data pass completely round the loop and return to the sender. A line tap attached to an Ethernet or ring network gives access to all the information it carries. For example, some large organizations employ a set of interconnected Ethernets carrying information of many different levels of sensitivity. The station logic attached to the Ethernet tap filters out messages addressed to each destination, but a small change to that logic can make it extract information for other destinations, giving access up to the highest level of sensitivity for the information in that network. Without additional precautions such as encipherment, today's local area networks make a very insecure environment.

Microwave radio and satellite systems similarly expose large quantities of data to easy access. Together with the increasing use of electronic mail in the future, this provides good opportunities for large-scale tapping of traffic.

In the past, the mere quantity of data would protect most of the secrets because the target of the enemy would be buried in uninteresting information. But since the source and destination of packets and messages travels with them in their headers, extracting information travelling to or from a target is not difficult. This target may be a terminal, a person or a computer process or port, depending on the level of addressing. Furthermore, unprotected messages can be selected according to their content, extracting messages which contain certain words or bit patterns for review by the enemy. It can be seen that modern technology increases rather than decreases the opportunities.

Improved technology of information processing has aided the protection of information by making good ciphers available in the form of single-chip devices and by allowing complex systems to be made more compact and thus given adequate physical protection. But the same technology is available to the enemy who can buy low-cost processing and storage in order to investigate systems and exploit their weaknesses.

1.4. THE NOTATION FOR ENCIPHERMENT

Codes and ciphers are part of the subject of cryptography which is the art of hiding information in a secret way so as to preserve its confidentiality. We shall only be concerned with ciphers so the operations that are carried out are called encipherment and decipherment.

The original purpose of ciphers is to conceal information while it is being transmitted. Stored information can be concealed in a similar way. But the security of information systems is much wider in scope than merely concealment and

includes the integrity of messages and the authentication of one communicating party by another. Wherever a high level of security is needed a technique based on cryptography can be found, so cryptography proves to be the fundamental tool for building secure systems. The ways in which ciphers can be used for these purposes form a main theme of this book. For the present, we need to know how to describe these basic operations in cryptography.

We need a notation for the two operations of encipherment and decipherment. The pictorial convention we shall use is shown in Figure 1.1. The two operations can be described as functions of two variables, or they can be described by an algorithm — a systematic procedure for calculating the result when the values of the variables are given.

The *plaintext* mentioned in the figure is the set of data before encipherment. The result of encipherment is the *ciphertext*. The result of applying decipherment to the ciphertext is to restore the plaintext. These words 'plaintext' and 'ciphertext' are relative to a particular encipherment — it could easily happen that the plaintext shown here is the result of a previous encipherment but for our purpose, since it enters into the encipherment function, it is called the plaintext. The capital letters D and E are reserved for the decipherment and encipherment functions respectively. In the mathematical notation we shall use, if x is the plaintext then

$$y = Ek(x)$$

the result of encipherment, and the inverse function

$$x = Dk(y)$$

expresses decipherment of the ciphertext to produce the plaintext. In both functions, the other variable k is the *key*.

The key is an essential feature of a cipher. In principle, if the function $y = E(x)$, without a key, were kept secret, this might serve to conceal the value of x but, if the secret of the function E is lost, nothing can be done to restore the usefulness of the cipher. An entirely new cipher is needed. Since the design and testing of a cipher is a very big task, its loss in this way would be expensive. Furthermore, all those who took part in the design and testing or who wrote programs for the encipherment and decipherment procedures would have to be trusted indefinitely. These secrecy problems are made much easier by employing a *key*, a parameter which, in effect, allows a large class of ciphers $y = Ek(x)$ to be defined. If a key is compromised (discovered by an enemy, or by someone untrustworthy) it can be changed and the cipher function can remain in use.

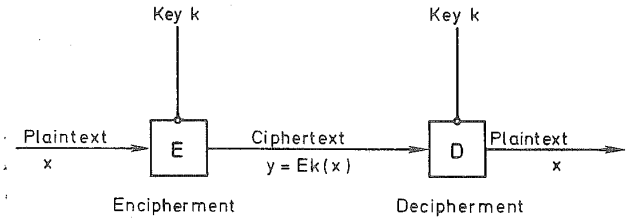


Fig. 1.1 Notation for encipherment and decipherment

Secrecy of the function or algorithm $Ek(x)$ as well as the key is a useful security measure, but, considering how many people must know its form, a secret function cannot be reliable by itself. It is the secrecy of the *key* that matters. This cipher must remain effective even if the enemy knows the function $Ek(x)$. When $Ek(x)$ is a *standard* cipher, like the Data Encryption Standard, everyone who is interested can find out the function.

The notation $Ek(x)$ is derived from one that has now been widely used and it probably started with k as a suffix $E_k(x)$ to denote that k is a parameter, which is kept constant for a number of encipherments. In the pictorial form of the notation it is necessary to distinguish between the two inputs to the boxes which denote the functions E and D. Therefore we use the small circle shown in Figure 1.1 to show which is the *key* input.

Other notations for encipherment have their uses. If complex expressions or lists are enciphered and when the procedure of decipherment is not shown explicitly, the notation $\{\ldots\}^k$ is often used, where the data or text in the curly brackets is enciphered with key k. This is the notation that has been used widely in discussions of authentication. We have preferred to keep a uniform notation throughout this work and $y = Ek(x)$ is our choice. If there are several keys in use these could be shown with suffices as k_{ab}, k_s etc, but in this book we put these symbols on the same line. Thus $Ekab(x)$ means that x is enciphered, using as key the value *kab*. This *kab* is similar to the 'identifier' in a programming language — it does not, in this context, mean that k, a and b are separate variables. All our key identifiers will begin with the letter k.

The domain of values from which the key k is taken is called the *key space*. In a typical cipher, the key might be 20 decimal digits, for example, which gives a key space of size 10^{20}. Alternatively the key might be expressed as a binary number of 50 digits with a size of key space 2^{50}. In other ciphers, the key is a permutation such as the letters A, B ... Z in some sequence, consequently the key space has size 26!

The notation $y = Ek(x)$ treats x and y also as single variables, which have values in a finite domain. Since x represents, in general, a block of text, a cipher of this kind is called a *block cipher*. For example, eight characters in ISO 7-bit code with parity make up a block of 64 bits. The cipher could be designed for a 64-bit block, but the redundancy of the ISO code makes the information content much less. If the characters are taken from text in a natural language such as English, the true information content of x is smaller still.

We also need a notation for the encipherment of messages of indefinite length. The single variable x is replaced by a string of variables. For this situation we write the cipher text as:

$$Ek(x_0, x_1, \ldots x_i, \ldots)$$

where x_0, x_1 ... is a string of variables whose size is given by the context. They could be parts of the message or its header, such as the source identifier or a sequence number. The simplest case is the one in which each variable x_i is a character with its value taken from a finite set of values known as the *alphabet*. For example the alphabet might be, literally, A, B, ... Z or it might contain the 256 different values of the octet (i.e. an 8-bit character). Ciphers which are designed to encipher a string of such characters with no theoretical limit to the length of

the string are sometimes called *stream ciphers*. Unfortunately, the terminology has developed haphazardly and there is no clearly understood and agreed definition of this term. It is usually employed when the alphabet is fairly small — the extreme case is the bit stream with an alphabet of 0 and 1. Obviously, the ciphertext for small alphabets must be calculated from more than just the current character or it becomes trivially simple.

The need for key distribution and management

When encryption is used to make data secure in communications, there must be prior agreement between the communicating parties about all aspects of the procedure. A cipher algorithm, and the method of using it and initializing it, must be agreed. The most difficult requirement is that a key must be chosen and made available at both ends of the communication path. Before the enciphered data flow over the line, the value of the key must make a similar journey. Keys can be enciphered using other keys but in the end at least one key has to be distributed by some other means. The cipher can therefore be regarded not as a creator of secrecy but as a means of extending the secrecy applied to the distribution of the key into the transport of a much larger volume of data enciphered with that key.

When encipherment is used for protecting stored data, key handling is easier because the encipherment and decipherment are carried out at the same location, so the key need not travel, but secrecy of the stored key is still important.

Choosing the key in the first place and making it available only to authorized users are aspects of *key management*. After encipherment methods have been decided, key management becomes a major task for system designers because the security of the whole system has then been concentrated on the keys. Chapter 6 is devoted to key management.

1.5. SOME USES FOR ENCIPHERMENT

Encipherment conceals the content of a message, except for those intended to receive it who, knowing the key, can decipher it. Thus it controls access to the message content, making it readable to those who know the key. If the ciphertext is made public and the cipher algorithm is well known, knowledge of the key is the equivalent of access to the message. Knowing the key also allows a person to construct another message so, under the right circumstances, a message can both be read and written. By using a 'public key cipher' it is possible to have separate control of reading and writing, as we shall see in chapter 8.

Without a knowledge of the key an enemy cannot usefully alter a message. He can probably alter the ciphertext but when that is deciphered it will read as a random pattern, because lack of the key prevents the enemy having control of its deciphered form. Assuming that the message has some redundancy, the authentic message produced with the aid of the key can easily be distinguished from the random pattern. In this way its authenticity can be proved to whoever knows the secret key. We discuss authentication in more detail in chapter 5. It is one of the principal uses of encipherment methods. These operations of *access control* and *authentication* can be applied to transmitted messages or stored files.

Together they sum up the various uses of encipherment but, like any generalizations they conceal the great variety of ways that ciphers can be used. To illustrate this, consider the protection of a claim to priority in an invention.

Before the institution of patents for inventions, if an inventor published his methods he lost the ability to exploit it before others could. Some inventors wanted to put their brainchild on record and establish their priority without revealing it to competitors. To do this they wrote a description and enciphered it, publishing the cipher and keeping the key secret. Later, when the invention became well known, others might claim to be its originator. Then the true inventor revealed the key so that anyone could verify that he published it at the earlier date.

In chapter 8 we describe a novel kind of cipher in which both sender and receiver have secret keys but neither sends his key to the other (page 224). This was first described in the context of 'Mental Poker' — a game of poker played between people who are all in different places and joined only by telecommunication channels. This is a vivid illustration of a use of encipherment in which secrecy and authentication are not the whole story. We can illustrate this by a simpler case, the tossing of a coin by two people at a distance, using only messages to link them.

The two people are Ann and Bill. Deliberate bias in their coin-tossing can be avoided by a convention in which Ann announces her call C_a with its value 'heads' or 'tails' and Bill similarly announces C_b then the outcome of the procedure is a head if the two calls coincide and a tail if they differ. The simple analysis of this 'game' shows that both Ann and Bill will choose between head and tail at random. If either shows a bias the other can improve their chance of winning.

If Ann announces her call first then Bill can arrange to win, and vice versa. If no trusted third person is available to receive the calls and adjudicate, the two parties can use encipherment in the following way.

Ann chooses one of two highly redundant statements such as 'My call for this game is heads' or 'My call for this game is tails', enciphers whichever phrase she chooses and keeps the key secret. Because of the redundancy she cannot find two keys that will generate either message from the same ciphertext. Of course there must be no other clues such as the length of the mesage. Ann and Bill exchange their enciphered calls. When Ann has received Bill's call she reveals her key and vice versa. If each finds the deciphered message to be one of the two phrases, the calls stand and the outcome is as defined. The use of encipherment authenticates the calls after they have been exchanged and ensures that they cannot be modified after learning the other's call.

A large part of the practice of data security consists of procedures (or protocols) using encipherment of which the coin-tossing procedure is just a simple example. The procedures range from the very simple to rather complex schemes such as those of key management described in chapter 6.

1.6. GENERAL PROPERTIES OF CIPHER FUNCTIONS

The most useful ciphers are those which do not expand the text. A block cipher for an n-bit block maps the 2^n different plaintext values onto the 2^n different ciphertext values. In effect, it is a permutation of these 2^n values, one which

varies with the value of key. The possible number of permutations is $(2^n)!$ which is usually larger than the key space, so many of the possible permutations do not happen. For all practical purposes the permutations seem to be chosen at random in a good cipher because, if there were obvious regularities, the cryptanalyst could exploit them.

For a given value of key k, if x ranges over all its possible values, $y = Ek(x)$ also ranges over all its possible values. On the other hand if x is held constant and k ranges over all its key space, the y values generated can be regarded as random choices. Some y values may not occur at all, others may occur more than once.

When there is no data expansion the identity $x = Dk[Ek(x)]$ implies another identity $x = Ek[Dk(x)]$. In other words, the encipherment and decipherment can be interchanged. This is illustrated in Figure 1.2 and it is not a trivial result. If there is data expansion then $Dk(x)$ *reduces* the size of the text and the second identity cannot be true. Bear in mind that, when encipherment and decipherment are exchanged, though they have all the formal properties of a decipherment/encipherment pair, some of their useful properties may disappear. Figure 1.2 is a trivial example based on a 3-bit character.

The type of function shown in Figure 1.2 is called a *bijection*. This kind of function maps a set of values into a set of equal size. Each value maps into just one corresponding value, both for the function and its inverse. This is a one-to-one mapping.

There is a special kind of cipher which is an *involution*, that is a function $f(x)$ which has the property that $f[f(x)] = x$. In the case of a cipher function which is an involution,

$$Ek[Ek(x)] = x$$

implies that $Ek(x) = Dk(x)$, encipherment and decipherment are identical. In the days when ciphers were produced by mechanisms a 'self-inverse' cipher was an advantage because the mechanism needed no adjustment between sending and receiving. Many examples exist of involutions as cipher functions.

The product of two involutions is not necessarily an involution, as Figure 1.3

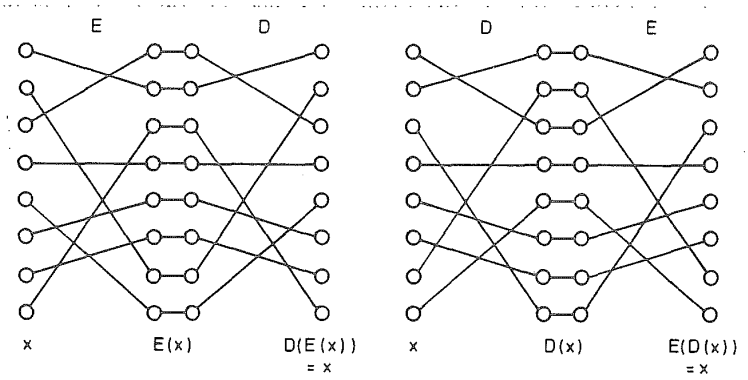


Fig. 1.2 Exchange of cipher and decipher functions

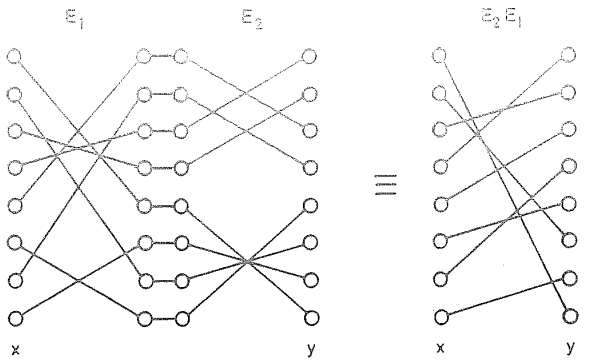


Fig. 1.3 The product of two involutions

illustrates. The inverse of such a product can easily be found if its components are known. If encipherment is

$$y = E_2[E_1(x)]$$

and both E_1 and E_2 are involutions, the decipherment is

$$x = E_1[E_2(y)]$$

A series of involution transformations can produce a secure cipher which is easily inverted, and this principle underlies the construction of the Data Encryption Standard described in chapter 3.

Chapter 2 CIPHERS AND THEIR PROPERTIES

2.1. INTRODUCTION

The known history of codes and ciphers is long and fascinating, dating back almost 4 000 years to the time when the Egyptians, possibly out of respect for the dead, used a hieroglyphic code for inscriptions on tombs. It is likely that the art of concealment of meaning of written communication goes back even further, though the Egyptian funerary inscriptions are amongst the earliest known examples. Many and various have been the techniques employed over the centuries. By today most of the 'classical' techniques are thought to be too weak for serious application; however, many of their basic principles can still be identified as forming part of modern techniques and therefore it is worth examining how some of them worked. This chapter aims to give a flavour of classical techniques without attempting to be exhaustive. For a comprehensive treatment readers should consult David Kahn's book *The Codebreakers*.[1] Another useful reference on classical techniques is the book *Cryptanalysis* by Helen Fouché Gaines.[2]

Concealment may take two forms, either the message is hidden in such a way that the very fact of its existence is obscured or the message is transformed so as to be unintelligible even though its existence is apparent. The first of these techniques has been called 'steganography' (literally: covered writing), whilst the second is known as 'cryptography' (literally: hidden writing); the literal distinction between these terms seems very fine, but their technical significance makes a rather useful distinction.

A steganographic method said to have been popular in classical times involved shaving the head of a slave, writing the message on the shaven head, waiting till the hair grew again and then despatching the slave to have his head shaved by the intended message recipient. This technique has little to commend it, least of all the slow rate of communication (unless early Greek hair grew much faster than does ours). Another method used pictures to carry hidden messages; Figure 2.1 illustrates an example. At first glance this seems to be just another drawing of a shrub; however, a concealed message is contained within the picture. Reading clockwise around the outline of the shrub (beginning at the trunk) and recording a '1' for each twig bearing a fruit and a '0' for each empty twig, we obtain a binary message. Enclosed regions are ignored — they only add to the pictorial effect — but in a more elaborate 'tree-structured' reading of the figure they could be used, adding 16 bits to its message.

Lord Bacon's method of steganography used two type founts, differing slightly in appearance. Effectively one fount stood for binary 0 and the other for binary 1, permitting encoding of letters of the secret message within a totally unrelated text. Much effort has gone into attempts to prove that Bacon was the true author

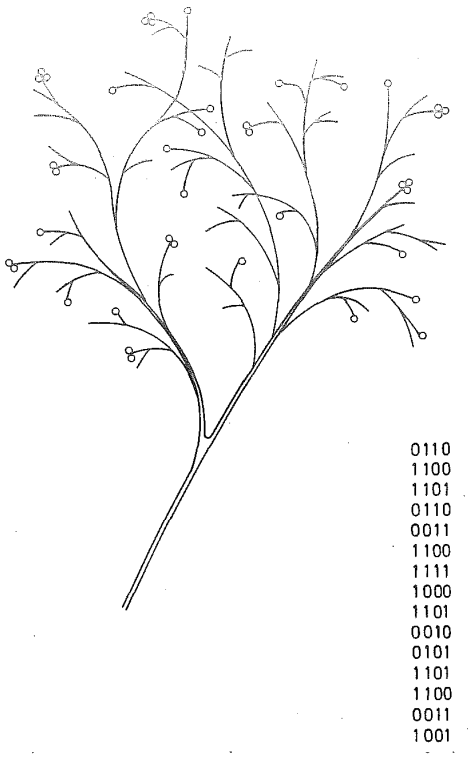


Fig. 2.1 A picture containing a hidden message — steganography

of the plays attributed to Shakespeare; one approach was to scan the first folios of these plays for messages hidden by Bacon's method. The famous American cryptologist, William Friedman, began his cryptographic career in a fruitless search for these messages at the Riverbank Laboratories, Geneva, Illinois. Under Friedman's leadership the Riverbank Laboratories became highly respected in cryptology but their owner, George Fabyan, never gave up his belief that the Baconian ciphers could be found in Shakespeare.

In yet again another steganographic method, messages have been expressed by patterns of goods in shop windows. Writing a message in invisible ink could be said to use steganography. The variations on the steganographic theme are endless. However, the nearest thing to a steganographic technique that is useful for modern data protection is that of filling the spaces between encrypted messages with meaningless garble when these messages are transmitted over communication links. The intention is to present the appearance of a link busy at all times and thus conceal the passage of messages; unlike real steganography, the existence of a communication link is evident, only the traffic level is concealed.

We shall find cryptography of far greater significance than steganography as we consider methods of achieving security of data. Here the important thing is so to change the message that its meaning is completely concealed from anyone

not entitled to read it. We therefore need some kind of message transformation and for this purpose two basic methods have been devised — codes and ciphers.

To encode a message its words or groups of words are looked up in a code book containing code equivalents which may be alphabetic or numeric; the book is arranged in sequential order of words or word groups. Replacement of the component parts of a message by their code equivalents constitutes the entire process of encoding. To decode an encoded message efficiently a second, matching code book is required, this time indexed according to the code elements. A brief example, based on the international radio 'Q' code, is given in Figure 2.2. Codes are often applied today in ordinary commercial correspondence in order to achieve message compression and economize on transmission costs, by using single-word equivalents to represent plaintext word groups. Many such codes have been devised and offered for sale; company letter headings often carry a list of codes used; such codes are public knowledge and can therefore not be used for message concealment.

Secret codes have held an honourable place in the techniques of general-purpose cryptography. They can be made more effective for this purpose by ensuring that multiple equivalents are listed in the code books, so that the code element used for a particular word or word group is not always the same, but is chosen at random among the alternatives. In this way an important word which frequently occurs does not leave a tell-tale frequency in the coded text. For the protection of computer data, codes are far from ideal, because the equivalent of a code book would make bulky storage demands and the process of look-up might be time consuming. The generation of an adequate code book is itself a formidable task. Protection of stored code tables, vital for security, could itself be difficult to achieve. The secure and rapid transport of bulky code tables to the locations where they are required is potentially a difficult problem, especially if frequent change of code is required in order to maintain security.

Unlike cryptographic codes, ciphers are not concerned with whole words or groups of words. In the past they have operated, generally speaking, at the level of the individual letter; modern cipher techniques applied in a computer context operate sometimes with big units of many characters, sometimes at the character or letter level, and sometimes at the level of the individual binary digit. Some of the classical techniques which have operated at the letter level are considered in the next section.

Before leaving the subject of codes, the important class of secrecy systems known as *nomenclators*[1] must be mentioned. These had their origin round about the year 1380 and at first consisted simply of code substitutions for important elements

Are you busy?	QRL	QRH	Does my frequency vary?
Does my frequency vary?	QRH	QRL	Are you busy?
Is my keying correct?	QSD	QRO	Shall I increase power?
Shall I increase power?	QRO	QRZ	Who is calling me?
What is the exact time?	QTR	QSD	Is my keying correct?
Who is calling me?	QRZ	QTR	What is the exact time?

Fig. 2.2 Brief extract from a two-part code book

such as names of places or people, the rest of the text being left in clear. After about 20 years the system was extended to include code substitutions for important word or letter groups. As an example, the phrase 'of the' or the fragment 'ing' might have been included. At the same time a letter-by-letter substitution cipher was introduced to apply to the portions of text not appearing in the code lists. It is noteworthy that the designers of this combination of code and cipher were sufficiently aware of the possibility of cryptanalysis by frequency counts that they included multiple equivalents (known as 'homophones') in their substitution cipher.

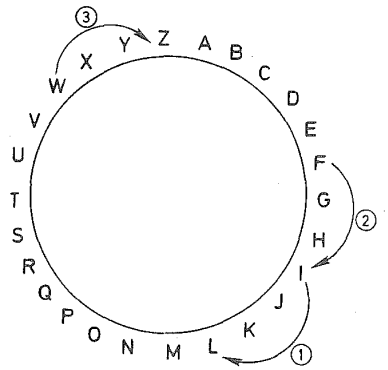
Nomenclators survived in popular use until the middle of the nineteenth century, when commercial codes were developed in order to economize on message lengths in telegraph transmission and improved understanding of cipher cryptanalysis led to the development of ciphers that were stronger than simple substitution.

The two basic components of classical cipher techniques are *substitution* and *transposition*. In substitution, letters are replaced by other letters, whilst in transposition letters are arranged in a different order.

2.2. SUBSTITUTION CIPHERS

The Caesar cipher

If we wish to replace letters of a message by other letters in a systematic fashion, then one of the simplest ways is to express the alphabet in cyclic form (Figure 2.3) and then select letters shifted by a specified number of places in a particular direction from each plaintext letter. An example of an enciphered message is given in the figure. Here the letter displacement is by three places to the right and the cipher corresponds to one credited to Julius Caesar. Another classical cipher of this kind is said to have been due to Caesar Augustus, Julius' nephew, who chose a letter displacement of one position. Such a cipher is extremely easy to operate;



Plaintext I F W E W I S H T O R E P L A C E L E T T E R S O F A ____
Ciphertext L I Z H Z L V K W R U H S O D F H O H W W H U V R I D

Fig. 2.3 Caesar's cipher, $c = p + 3$ modulo 26

```
                GDUCUGQFRMPCNJYACJCRRCPQ
                HEVDVHRGSNQDOKZBDKDSSDQR
Plain -         IFWEWISHTOREPLACELETTERS
                JGXFXJTIUPSFQMBDFMFUUFST
                KHYGYKUJVQTGRNCEGNGVVGTU
Cipher -        LIZHZLVKWRUHSODFHOHWWHUV
                MJAIAMWLXSVITPEGIPIXXIVW
                NKBJBNXMYTWJUQFHJQJYYJWX
```

Fig. 2.4 Breaking a cipher of the Caesar type by searching twenty-six keys

the alphabet written on two concentric discs displaced by the required amount makes a useful tool for encipherment and decipherment. The cryptologist may even attempt to make things more difficult for the cryptanalyst by changing the amount of the displacement from time to time. If, in Julius Caesar's cipher, the letters A, B C ... are given numbers 0, 1, 2 ... the process of encipherment can be expressed as $c = p + 3$ (modulo 26), where the addition requires 26 to be subtracted if the result exceeds 25.

Unfortunately such a cipher, which must surely have been re-invented by countless schoolboys, is ridiculously weak. If the amount of the displacement is known, then there is no secret and the plaintext is readily retrievable; even if the displacement is not known, then it can be discovered very easily. All one need do is to write out the ciphered message followed by all its possible displacements by from one to twenty-five letters. One of the transformed messages will correspond to the plaintext and will be obvious to the reader from the meaning conveyed, assuming that there is adequate redundancy in the message (a plaintext message consisting entirely of arbitrary characters would present a problem). In Figure 2.4 we give an example of what is involved; the displacement here is of three letters, as in the Caesar cipher. In Figure 2.4, and indeed in all subsequent figures in this chapter, spaces between words are omitted; they are ignored in the cipher operation. This convention is customary when simple ciphers are used. They could have been treated as a twenty-seventh letter and enciphered like all the others, but their high frequency weakens any simple cipher.

The technique of trying all possible displacements to solve a Caesar-type cipher is effectively a key search; the displacement is the key and possible values are tested until the right one is found by inspecting the 'plaintext' produced in each case.

Monoalphabetic substitution

Somewhat greater security may be obtained if the letter substitution is carried out with a 'jumbled alphabet'. In Figure 2.5 we show the message of Figure 2.3

```
Plain       ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher      DKVQFIBJWPESCXHTMYAUOLRGZN

Plaintext   IFWEWISHTOREPLACELETTERS
Ciphertext  WIRFRWAJUHYFTSDVFSFUUFYA
```

Fig. 2.5 A monoalphabetic substitution

encrypted using such a method. Effectively we are making the displacement of the cipher letter depend on the particular plaintext letter. This technique is known as a monoalphabetic substitution because one jumbled alphabet is used. Whereas the number of possible displacements and therefore the number of possible encipherments under a Caesar-type cipher is 25, the number of different jumbled English alphabets is 26! or about 4×10^{26}. With such a large number of possibilities monoalphabetic substitution is seemingly a very strong cipher system. Sadly its apparent strength is a delusion. Two reasons can be found for the weakness: the statistical clues given to the cryptanalyst are very strong and the alphabet can be discovered piece by piece. Even if the alphabet used has many wrong letters, the cipher can be read, like a badly transmitted telegram, and the wrong letters can be corrected. In a good cipher, getting the key nearly right reveals nothing of the plaintext.

The monoalphabetic substitution illustrates the fallacy of relying on double encipherment for increased strength. Double encipherment with different alphabets is equivalent to single encipherment with an alphabet formed by applying the second substitution to the first alphabet.

Cryptanalysis of a message enciphered with a monoalphabetic substitution depends on the fact that each plaintext letter is always transformed into the same ciphertext equivalent. In natural language, letter frequency tables can be set up which show the relative order of use of letters of the alphabet. For English text the order is generally that shown in Figure 2.6 (after Gaines). Depending on the text from which the frequency tables are created we may find small variations

	%		%		%
E	13.1	D	4.1	G	1.4
T	9.0	L	3.6	B	1.3
O	8.2	C	2.9	V	1.0
A	7.8	F	2.9	K	0.4
N	7.3	U	2.8	X	0.3
I	6.8	M	2.6	J	0.2
R	6.6	P	2.2	Q	0.1
S	6.5	Y	1.5	Z	0.1
H	5.9	W	1.5		

Letter frequencies in English text

TH	THE
HE	AND
AN	THA
IN	ENT
ER	ION
RE	TIO
ES	FOR
ON	NDE
EA	HAS
TI	NCE

The ten most frequent digrams and trigrams in English text

Fig. 2.6 Frequencies of letters, digrams and trigrams in English text

in the ranked order. For cryptanalysis, the letter frequency count of the enciphered message is calculated; from this it should be apparent which are the most frequent letters and which are the least frequent; the middle-frequency letters convey the least information in the process of cryptanalysis. By trial and error the cryptanalyst tries out possibilities for the plaintext letter equivalents, relying on redundancy in the plaintext to indicate when he is hot on the scent. Aid may be obtained from tables of digraph and trigraph frequencies in English, which will show what letters are most likely to be adjacent in text. Figure 2.6 also shows a few of the most frequent English digraphs and trigraphs. If the cryptanalyst is not even sure that he is dealing with an English enciphered message, then the frequency count, if the ciphertext is long enough, may give him a clue. For example the letter 'Q' ranks very low in English and German, but rather higher in French and Italian; the vowel 'O' ranks relatively high in English, Italian, Spanish and Portuguese, but rather lower in French and particularly low in German. Figure 2.7 shows letter frequency profiles for the more frequent letters of English, German and Italian (the spot values are joined together to assist in reading the profiles). For the four most frequent letters in each language the profile is very distinctive; beyond this the profiles rapidly become indistinguishable. From such

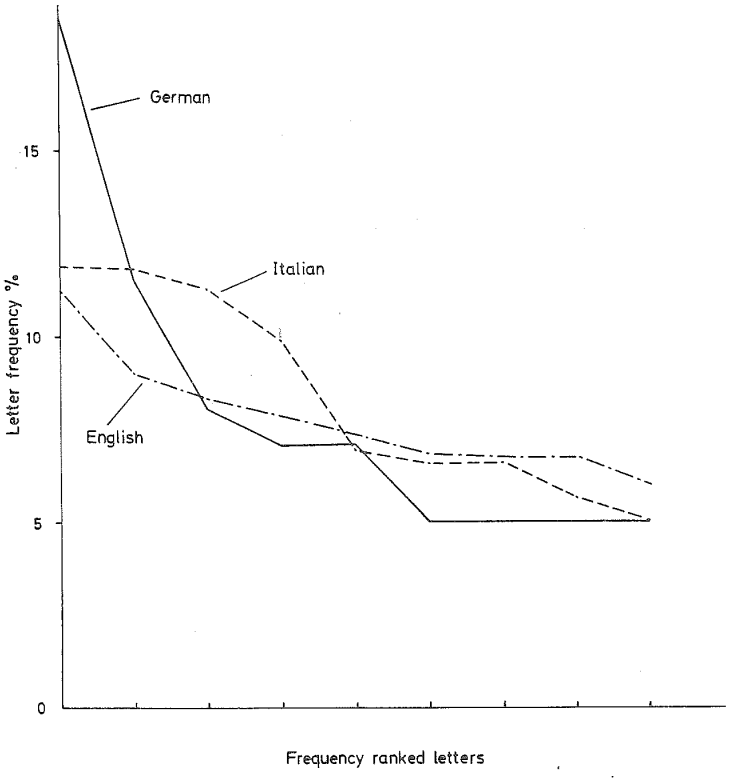


Fig. 2.7 Letter frequency profiles for English, German and Italian

indications of the letter frequency distribution it may be possible to make an informed guess at the language of the plaintext even before the message has been cryptanalysed.

Polyalphabetic substitution

Realizing this weakness of monoalphabetic ciphers the cryptologists went one better — they created polyalphabetic ciphers. Here there is not one jumbled alphabet for use as a substitution table, but a number of them which are used in succession on the successive plaintext letters. When the last alphabet has been used, we start again with the first. Because each letter of the plaintext is no longer always represented by the same letter in the ciphertext, it is not possible to take a simple frequency count and deduce the shape of the substitution — more subtlety is required. It is of great help to the cryptanalyst to find out how many alphabets are in use; for this there are various techniques available. One of them depends on the frequent repetition of letter groups in text; the letters *the* are very common in English. In a sufficiently long text there is a very good chance that corresponding repeated letter groups may be found in the ciphertext, and they will sometimes be located at intervals of some multiple of the number of alphabets in use. These can be found by scanning the ciphertext and, from their spacing, intelligent guesses made as to the number of alphabets. A short text enciphered with five different alphabets is shown in Figure 2.8; note the repeated letter groups in the ciphertext occurring at a multiple of five letters. This method of determining the number of alphabets in use was made public in 1863 and is due to Kasiski.[1] Once the number of different alphabets is known, the cryptanalysis is simple if the text is long enough. All one does is to take frequency counts of the ciphered letters enciphered under each individual alphabet. The process then proceeds by trial and error as in the case of monoalphabetic substitution.

In both monoalphabetic and polyalphabetic substitution the common secret information that must be shared by sender and receiver of an enciphered message is the mixed alphabet or alphabets. This information forms the cryptographic key, which has to be distributed to both sender and receiver. A compact form of key is more convenient. For this purpose the alphabet can be generated from a key word as demonstrated in Figure 2.9. A key word of ten or twelve letters, in which each letter occurs once, might be suitable. In creating the mixed alphabet the key

```
Plaintext  THEREISNOOTHERMATTER
   Cipher  MBJSRBMSPBMBJSZTNYFE
           <--------->
           10 letters
```

Fig. 2.8 A polyalphabetic substitution with five alphabets

```
Plain   ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher  EXHAUSTINGBCDFJKLMOPQRVWYZ
```

Fig. 2.9 Method of generating a substitution alphabet using a key word

word is first written, then all the remaining letters in their normal order. To communicate such a mixed alphabet to a correspondent all that is necessary is to send the key word. Evidently there is economy of data transmission, which might be important if several mixed alphabets are to be communicated. Another advantage is that the key word may be remembered easily, useful if it is being carried in someone's memory. Note that at the end of the sequence some of the letters of the cipher alphabet are unchanged from the plain alphabet. This is sometimes avoided by an additional cyclic shift of the alphabet, as in the Caesar cipher.

The Vigenère cipher

Key words play an important part in a particular form of polyalphabetic cipher known as the Vigenère after its inventor and first published in 1586.[1] Here the multiple alphabets are not jumbled but displaced from the normal with a cyclic shift (compare the construction of the Caesar cipher). Application of the Vigenère cipher is facilitated by using a tableau, in which all the possible displaced alphabets are tabulated. Figure 2.10 shows such a tableau.

In a Vigenère encipherment the key word determines which displaced alphabet is used to encipher each successive letter of the plaintext message. This process can also be expressed as a modulo 26 addition of the letters of the key word (repeated as many times as necessary) into the plaintext letters. In Figure 2.11 we show a short section of plaintext enciphered with the key word 'cipher'. The process of modulo 26 addition can be carried out conveniently by means of a device

```
  ABCDEFGHIJKLMNOPQRSTUVWXYZ

A ABCDEFGHIJKLMNOPQRSTUVWXYZ
B BCDEFGHIJKLMNOPQRSTUVWXYZA
C CDEFGHIJKLMNOPQRSTUVWXYZAB
D DEFGHIJKLMNOPQRSTUVWXYZABC
E EFGHIJKLMNOPQRSTUVWXYZABCD
F FGHIJKLMNOPQRSTUVWXYZABCDE
G GHIJKLMNOPQRSTUVWXYZABCDEF
H HIJKLMNOPQRSTUVWXYZABCDEFG
I IJKLMNOPQRSTUVWXYZABCDEFGH
J JKLMNOPQRSTUVWXYZABCDEFGHI
K KLMNOPQRSTUVWXYZABCDEFGHIJ
L LMNOPQRSTUVWXYZABCDEFGHIJK
M MNOPQRSTUVWXYZABCDEFGHIJKL
N NOPQRSTUVWXYZABCDEFGHIJKLM
O OPQRSTUVWXYZABCDEFGHIJKLMN
P PQRSTUVWXYZABCDEFGHIJKLMNO
Q QRSTUVWXYZABCDEFGHIJKLMNOP
R RSTUVWXYZABCDEFGHIJKLMNOPQ
S STUVWXYZABCDEFGHIJKLMNOPQR
T TUVWXYZABCDEFGHIJKLMNOPQRS
U UVWXYZABCDEFGHIJKLMNOPQRST
V VWXYZABCDEFGHIJKLMNOPQRSTU
W WXYZABCDEFGHIJKLMNOPQRSTUV
X XYZABCDEFGHIJKLMNOPQRSTUVW
Y YZABCDEFGHIJKLMNOPQRSTUVWX
Z ZABCDEFGHIJKLMNOPQRSTUVWXY
```

Fig. 2.10 A Vigenère tableau

```
Plaintext    THISPROCESSCANALSOBEEXPRESSED
Keyword      CIPHERCIPHERCIPHERCIPHERCIPHE
Ciphertext   VPXZTIQKTZWTCVPSWFDMTETIGAHLH
```

Fig. 2.11 A Vigenère encipherment

called a 'Saint-Cyr' slide (invented by Kerckhoffs and named after the French military academy), rather like a slide rule (Figure 2.12). The same device carries out the modulo 26 subtraction needed for decipherment.

An attack on a Vigenère-enciphered message can be carried out by many of the methods applicable to polyalphabetic ciphers in general. One method of attack is particularly powerful against the Vigenère cipher — the 'probable word' attack. The cryptanalyst chooses a word which he considers is likely to be in the plaintext message and then carries out modulo 26 subtraction of that word from the ciphertext in all possible locations, i.e. a scan is made of the entire ciphertext. If at any position the result of the subtraction yields a word of natural language or a fragment of such a word, then it is very likely that the key word or part of it has been discovered. The whole message can then be deciphered easily. In Figure 2.13 we take the ciphertext of Figure 2.11 and show how the key word 'cipher' is revealed by guessing that the word 'process' is in the plaintext. Only when 'process' is subtracted from the ciphertext in the right position do we see fragments making up a plausible key word. A procedure like this employs the strong redundancy of natural language texts.

A particular form of Vigenère cipher uses a 'running key'. For this purpose a text is chosen by the sender and notified secretly to the receiver; the text must be something that is readily available, for example, part of a standard work of reference or a well-known novel. To encipher the message it is combined letter by letter by modulo 26 addition with the chosen text; decipherment is carried

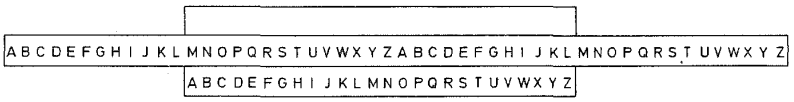


Fig. 2.12 The Saint-Cyr slide

```
Ciphertext   VPXZTIQKTZWTCVPSWFDMTETIGAHLH
     —       PROCESS
Result       GYJXPQY
     —         PROCESS
Result         AGLREYS
     —           PROCESS
Result           IOFGMSB
     —             PROCESS
Result             KCUOGBH
     —               PROCESS
Result               ERCIPHE ← Key word revealed
```

Fig. 2.13 A probable word attack on a Vigenère ciphertext

out by modulo 26 subtraction of the same text. This method is a favourite with new inventors.

With a natural language text as the encipherment key the cipher may not be secure against attack, but if the key text were a truly random series of characters, absolute security against cryptanalysis would be assured. The problem for the communicators is to establish the same random series at both ends of the communication channel. The channel which carries the key must carry as much data as the enciphered channel. The use of a pseudo-random generator at the two ends may be an acceptable alternative, providing that the pseudo-random cycle is very long indeed. This point is considered again in the context of the Vernam cipher.

It is generally recommended that a running key be used only once for encipherment, because certain cryptanalytic attacks can take advantage of its repeated use. Where two cipher texts are enciphered with the same key and are in phase, they are said to be enciphered 'in depth'. If the start points are not known it is possible to make use of a property known as the 'index of coincidence' in order, when shifting one ciphertext against the other, to determine when the two ciphertexts are in depth. The index of coincidence is a concept due to William Friedman, who defined it in connection with solutions[3] of the Vogel and Schneider ciphers. The concept makes use of letter frequency distribution in natural language. If two streams of letters are made up of totally random sequences (with equal probability for all letters), then the probability of any two corresponding letters being the same is $\frac{1}{26}$ or about 0.0385. If, on the other hand, the letters in the two streams obey the natural language letter frequency distribution (e.g. if they are meaningful plaintext streams), then the probability of coincidence in any letter pair from the two streams is about 0.0667 (for English text). If two ciphertexts are examined for coincident letter pairs, then the probability of coincidence will be about 0.0385, as for the random streams, unless they are enciphered with the same running key and are in phase, when the probability of coincident letters will be about 0.0667. Given amounts of ciphertext sufficient to establish the index of coincidence reasonably accurately, the difference in the two measures is sufficiently pronounced for the condition of encipherment in depth to be detected.

2.3. TRANSPOSITION CIPHERS

Transposition ciphers aim to hide the contents of a message by taking the individual characters or bits and rearranging their order. For decipherment an inverse process of unscrambling takes place. The word 'transposition' is commonly used for this kind of operation, though 'permutation' might be better.

A very early cipher based on transposition was the Greek 'scytale' (literally: staff). To operate this a narrow strip of writing material, such as parchment, was wrapped spirally around the staff. The message was then written in rows of characters longitudinally along the staff. Unwrapped, the strip appeared to carry meaningless sequences of characters. Only by carefully wrapping the strip around a staff of equal diameter would the plaintext message be apparent. The method is neat but not secure because the relation between strip width and staff diameter allows only a small key space.

Simple transposition

The simplest possible transpositions take successive letter groups and rearrange each in a regular fashion. For example the letters of a message might be taken in groups of five and rearranged in the order 41532; this sequence constitutes the encipherment key. Figure 2.14 shows a message enciphered in this way. A frequency count taken on such a message should show the letters occurring in their normal places in the ranking, suggesting that a transposition and not a substitution has taken place. Solution of a transposition cipher of this kind is on a par with the kind of anagramming which is carried out by crossword enthusiasts.

A convenient way to express the permutation in easily memorable form is by a key word. This operation is illustrated in Figure 2.15 where the key word is 'computer'. The letters of the key word are numbered according to their alphabetical order — 14358726. The plaintext letters are then written in rows of eight letters beneath the keyword, filling in the final row, if that is incomplete, with null characters. As a general rule, the user of a transposition cipher based on a letter array would be well advised to use garble rather than unusual letters standing as nulls. The use of a string of unusual letters helps cryptanalysis, because the cryptanalyst can find column and row arrangements bringing together these unusual letters. To create the ciphertext, each row is read out in succession in the order of the letters of the key word. Given that a message has been trans-

```
Plaintext    THESIMPLESTPOSSIBLETRANSPOSITIONSXX

Key                  4 1 5 3 2

                     S T I E H
                     E M S L P
                     S T S O P
                     E I T L B
                     S R P N A
                     T O I I S
                     X O X S N

Ciphertext   STIEHEMSLPSTSOPEITLBSRPNATOIISXOXSN
```

Fig. 2.14 A simple transposition cipher

```
Plaintext    ACONVENIENTWAYTOEXPRESSTHEPERMUTATION

Key word             C O M P U T E R

                     1 4 3 5 8 7 2 6

                     A N O V I N C E
                     E W T A O T N Y
                     E R P E T S X S
                     H E P R T U E M
                     A O I N Z Z T Z

Ciphertext   ANOVINCEEWTAOTNYERPETSXSHEPRTUEMAOINZZTZ
```

Fig. 2.15 Transposition based on a key word

posed in this way, the cryptanalyst can count the number of letters in the message and find the factors of this number. He can guess which of these factors was a likely length for a key word. Writing the ciphertext in rows of the width of this word, he would then attempt to re-order the columns in such a way as to make the text meaningful in all rows. A probable word helps the cryptanalyst because its letters do not get distributed very far and can be found and used to determine the transposition or part of it. If the word 'permutation' is thought to be present, the underlined letters are found. The same transposition in PER and ION establishes which E to use and the size of the permuted arrays.

The Nihilist cipher

Slightly more complex is the operation of the 'Nihilist' cipher which is a combination of columnar and row transposition. The plaintext is again written in rows under a key word, but this time the key word is also written vertically at the side of the columns (Figure 2.16). Ciphertext readout takes place row by row as in the previous example, but with the order of row selection being determined by the key word written vertically alongside; the effect of this process is to spread the mixture of plaintext letters over a domain equivalent in size to the square of the size of the key word. The key word is repeated as many times as necessary in the vertical dimension, the transposition process being taken separately for each key word repetition.

A cryptanalytic attack on a Nihilist cipher depends on column and row rearrangement, with much trial and error in seeking to build up words, aided by intuition. To program this kind of attack, use can be made of digraph and trigraph letter frequencies when testing trial rearrangements.

An alternative method of operating this kind of cipher is to write in the plaintext according to the sequence controlled by the key word and to read out the ciphertext according to some complex order, such as diagonally and in opposite directions for alternative diagonals, as also shown in Figure 2.16.

```
Plaintext    NOWISTHETIMEFORALLGOODMEN

                   L E M O N
                   2 1 3 5 4

             L 2   O N W S I
             E 1   H T E I T
             M 3   E M F R O
             O 5   L A L O G
             N 4   D O M N E

Nihilist Ciphertext   HTEITONWSIEMFRODOMNELALOG
Diagonal Ciphertext   ONHETWSEMLDAFIITRLOMOOGNE
```

Fig. 2.16 A Nihilist transposition and variant read-out

2.4. PRODUCT CIPHERS

We have now seen how some of the simpler forms of substitution and transposition ciphers operate and it is evident that they are not secure. Therefore, ciphers of this kind cannot be considered seriously for protecting data, though they may once have served a useful purpose.

Cryptanalysis of simple ciphers depends on the use of letter, digraph and trigraph frequency counts and on the ability to rearrange letter groups easily, looking for patterns of plaintext. How much more complex would the task of cryptanalysis be if substitution and transposition techniques were used in combination? Because of the substitution, it would no longer be possible to rearrange the rows and columns of a matrix by inspection. Because of the transposition, digraph and trigraph counts would be broken up.

Attractive though a product cipher might be from the point of view of cryptographic strength, it is too complex for human effort, unaided by a machine. The risk of making a mistake is too high and the encipherment and decipherment processes are slow. To use ciphers of this complexity requires cipher machines.

2.5. CIPHER MACHINES

Cryptographers have used mechanical aids of one kind or another for centuries; these began with simple devices like the Saint-Cyr slide and developed into very complex electromechanical systems by the time of the Second World War, before electronics took over.

The Jefferson cylinder

An interesting cipher device is credited to the American president, Thomas Jefferson, who in the 1790s developed a mechanism for applying polyalphabetic substitution. This consisted of thirty-six discs, each of which carried a jumbled alphabet on its periphery, which could be mounted on a common axis (Figure 2.17). The discs were numbered so that their order on the axis could be specified and agreed between communicators. The disc order constitutes the key; the number of different ways in which thirty-six discs can be arranged is of the order 10^{41}, therefore the key space is very large. In operation a message would be set up on the device by rotating the discs until the message letters stood on the same row against an index bar. To read out the ciphertext the bar could be moved to any of the other twenty-five positions around the cylinder; the row indicated would then be read out as the ciphertext. It is immaterial which row is chosen,

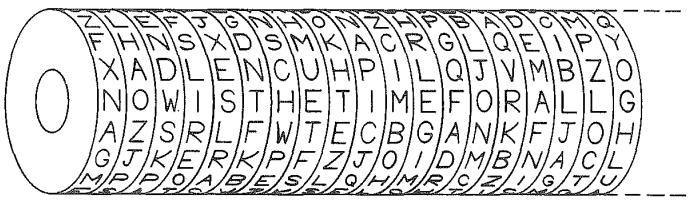


Fig. 2.17 The Jefferson wheel cipher device

because the intended recipient of the enciphered message would simply set up the row of ciphertext and then scan round the cylinder until a meaningful row of letters was found. It appears that this cipher was not used seriously until the 1920s, when it seems to have been re-invented and used by the US Navy for several decades.

The Wheatstone disc

Of slightly later date was the concentric disc type of cryptograph, originally invented by Wadsworth in 1817, but developed by Wheatstone in the 1860s. This type of device is effectively a polyalphabetic substitution cipher, but the way in which the various alphabets are used depends on the plaintext letter sequence. Two concentric discs carry the letters of the alphabet around their peripheries. Wheatstone's outer disc had twenty-six letters in alphabetical order plus space, whilst the inner disc had just twenty-six letters in random order (Figure 2.18). Two pointers were provided, geared together in such a way that when the outer had traversed twenty-seven letters so had the inner on its scale of twenty-six; if the pointers were coincident at the outset, after one complete revolution of the outer pointers the inner had moved by one revolution plus one letter, thus separating the pointers by one letter position. For encipherment the outer pointer was moved successively to the letters of the plaintext (moving always clockwise and including space as a character), whilst the ciphertext letters were indicated

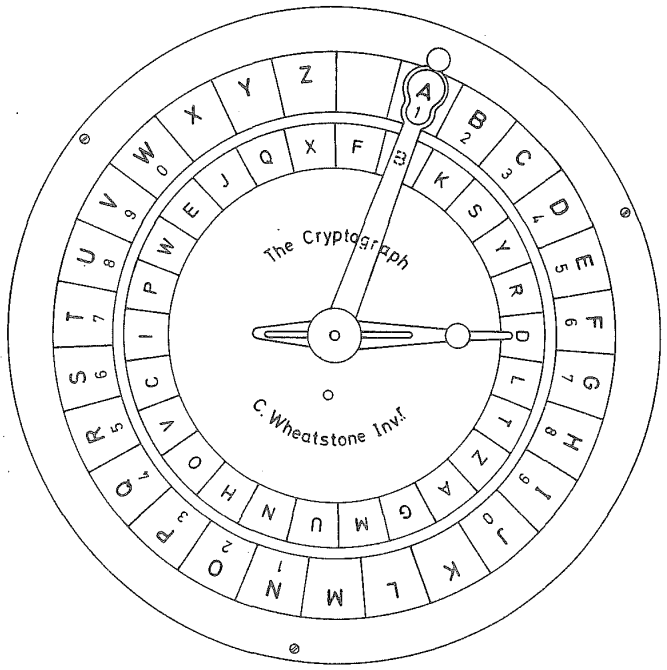


Fig. 2.18 The Wheatstone disc cipher device

by the inner pointer. The cipher is of the polyalphabetic substitution type with a change of alphabet at least at each word boundary (because of the space) and also within a word wherever successive characters of the plaintext are in reverse alphabetical order (or repeated). The cipher has the interesting property that the ciphertext representing a particular word is influenced by the preceding plaintext. This feature has properties similar to those of 'chaining', to be defined in chapter 4; chaining is of great importance in present-day applications.

Rotor machines, the Enigma

A very important class of cipher machines is based on various rotor designs in which rotors with internal cross-connections are used to achieve substitution with a continually changing alphabet. The internal links have been expressed both by pneumatic and electrical connections. Fibre optics came too late for the rotor machines.

One of the most famous polyalphabetic substitution machines was the Enigma (wiring shown schematically in Figure 2.19), a development from a Dutch invention dating from about 1920 and used by the German forces in the Second World War. This was a machine in which three, or in later machines four, rotors turned on a common shaft. Each rotor had twenty-six positions. The mode of rotation was similar to the action of an odometer with the rightmost rotor, R1, moving by one position for each letter enciphered and each rotor stepping the one to its left as it passed a certain position. Unlike a standard odometer the rotor, R2, in the middle stepped on itself by one additional position as it communicated a stepping action to the next rotor. Rotor R2 rested for only one character on its 'stepping' position and the effect was to give a three-rotor machine a cycle of $26 \times 25 \times 26$ characters. The four-rotor machine had the same cycle because its left-hand rotor did not move — no messages were long enough to make this matter.

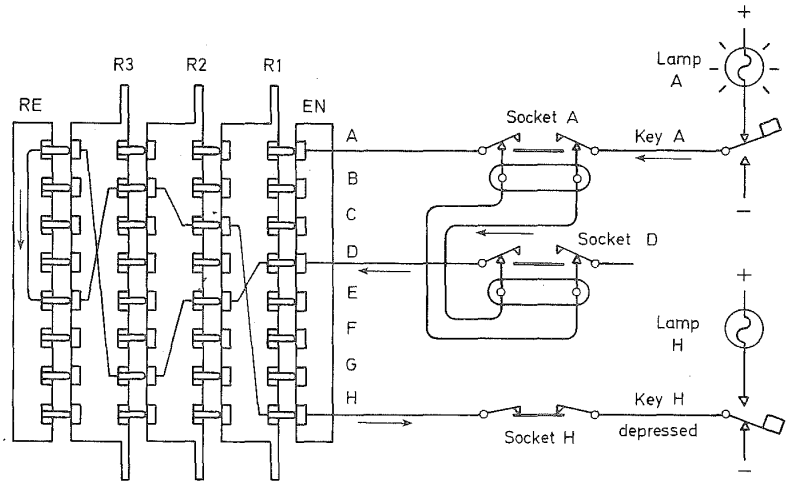


Fig. 2.19 Schematic of the Enigma cipher machine

Each Enigma rotor had twenty-six electrical contacts on each face, joined by internal wiring and the wiring pattern was different for each rotor. The fixed commutator, RE, to the left of the slowest-moving rotor produced a reflection of the electrical circuit back through the moving rotors by a different path. To make good contacts, one face of each rotor had sprung pins, as the figure illustrates.

When the original Enigma machine was adopted by the German forces, a plugboard was added to it. This had twenty-six sockets labelled A to Z and a number of double-ended cables were provided to join them in pairs. For most of the time ten cables were used, leaving six sockets unplugged. The plugboard was wired between the set of rotors and the set of keys and lamps. As shown in the figure for the two sockets A and F, the effect of a cable and its two plugs was to transpose one pair of connections that otherwise went straight through.

The figure shows only eight connections to represent the twenty-six connections of the actual machine. Depression of key H is shown, which disconnects that key from the lamp H and connects it, via the plugboard, to pin H on the fixed-entry commutator EN, since socket H is not used in this case.

Lamp A is connected through its key (undepressed) to socket A which is plugged to socket D. The effect of the cable is to transpose connections A and D. In the position of the rotors shown it happens that D and H are joined together so the connection to the negative supply at key H results in lighting lamp A. The plugboard permutes the connections from lamps/keys to EN and the permutation happens to be an involution (A to D implies D to A, as the figure shows). There is no logical reason why it should be an involution and late in the war a twenty-six position switch, the Enigma Uhr, was introduced which removed this restriction and allowed the plugboard setting to be changed each hour.

The figure shows the eight connections laid in a row instead of a ring of twenty-six. Rotation of any rotor completely changed the permutation of the twenty-six wires. The machine provided a sequence of $26 \times 25 \times 26$ substitutions before it repeated. In the fixed commutator, RE, on the left are thirteen connections. Since the key H operates lamp A it follows that, at the same rotor position, key A would operate lamp H, therefore each substitution is an involution — and this does not depend on how the plugboard is wired. It is a consequence of the thirteen wires in RE which themselves define an involution that is transformed by the rest of the equipment. It is also clear that no letter can be enciphered into itself because the electrical connections prevent that happening.

It was possible to replace Enigma rotors on the shaft from a collection or 'basket' of several different rotors. The initial setting of the machine required the selection and installation of rotors with their ring settings and then setting up the plug board. These operations constituted the current encipherment key. For each message, the initial rotor positions were chosen by the operator and transmitted in the message header. This 'indicator' was the weak point exploited in cryptanalysis. The German forces changed the method twice.

Operationally the Enigma machine was inconvenient. It needed at least two people, one to type the text on the keyboard and the other to read the lamps and write down the enciphered message. A printer attachment was developed but not widely used. There was also an extension attachment for the lamps so that only one person need see the plaintext.

It is now well known that, during the Second World War, messages enciphered

with the Enigma machine were regularly cryptanalysed, first in Poland, then in France and finally, on a large scale and with great effect, at the Government Code and Cipher School at Bletchley Park.[4]

Printing cipher machines

With the invention of the typewriter in the latter part of the nineteenth century it seems natural that efforts should be made to develop cipher machines in which the plaintext could be typed on a keyboard and the ciphertext produced directly by a printing mechanism in the machine. The first known machine of this type was due to De Viaris in the 1880s. Many other machines with this property followed, some of which were also able to transmit ciphertext directly onto a communication link. A notable example of a printing cipher machine was the celebrated M-209 machine of Boris Hagelin, used in large numbers during the Second World War. A full description of the M-209 is given by Beker and Piper.[5] The M-209 was only one of the very many different cipher machines invented by Hagelin.

On-line cipher machines take the automatic handling of text one stage further than the printer. In these, the plaintext is typed on a machine which enciphers it and transmits the ciphertext (via a telegraph circuit or a radio-telegraph channel) to the receiver. Here, the same type of machine deciphers the received text and prints it in clear form. No human handling of the ciphertext is needed.

One of the most highly developed of the on-line telegraph cipher machines was produced by Siemens and Halske from the early 1930s to 1945. The machine was called type T52 and was built around a standard electric teleprinter. It was probably also called the Geheimfernschreiber or Geheimschreiber. It was used very effectively, mainly on telegraph circuits, in the Second World War. In its simplest form this machine used 10 wheels, each wheel stepped on by one place for each letter sent. The sizes of wheels were 47, 53, 59, 61, 64, 65, 69, 71 and 73, relatively prime, potentially giving a repeating cycle of length about 9×10^{17}.

The wheels produced letter substitution on the 5-bit Baudot code by a combination of running-key addition from five wheels and permutation of the five code elements by five other wheels. The key setting of the machine was at first a plug and jack field for permuting the ten channels from the cams on their way to the encipher/decipher circuits. Later versions of the T52 produced an irregular movement of the ten wheels by stepping the wheels according to logical functions (derived by magnet circuits) of the outputs of the cams, taken from a different place on the cam. A further complication was that the ten binary encipherment channels, after permutation according to the key, were subjected to linear logical operations (XOR) so that each channel to the encipher/decipher circuits depended on four cam outputs. A detailed study of the development of the T52 machine has been made by Davies.[6]

Modern cipher machines

A wide range of cipher machines is currently available from many manufacturers. These machines are mostly communications oriented, provided with appropriate interfaces for connection to modems and communication lines. Their design details

are usually secret, even to customers who purchase them, the design being a commercial property of their manufacturers. Whether secrecy of design is desirable in cipher machines is a moot point. Should the security strength of the system depend on the secrecy of the encryption algorithm? If this were the only element on which security depended, then clearly the whole integrity of the protected communication would depend on preventing an intruder from discovering the details of the algorithm. It is realistic to assume that an intruder may be able to steal a machine and study it. Any machine worthy of consideration can be expected to possess a large key domain from which a secret key can be chosen; this is the other component in attaining adequate strength. As we saw in connection with monoalphabetic substitution ciphers, the size of the potential key domain is not necessarily a good indication of cryptographic strength.

Substitution in modern ciphers

A substitution can replace a 'word' of n bits by another word either as a mathematical (logical or arithmetical) function or using an arbitrary function expressed in a table of values.

We shall later meet examples of mathematical functions, such as those based on modular arithmetic. Addition, modulo 26, provides the Vigenère substitution. In chapters 8 and 9 exponential functions with very large moduli are used. However, the simplest mechanism for a cipher is an arbitrary table of values.

As an example consider an 8-bit substitution. There are 256 different values of the independent variable and for each value the 8-bit output must be specified. This can be done in 256 bytes of memory, which is a small expenditure in a micro-based encipherment device. In a programmed form, it usually requires two or three instructions and of the order of 1 μs of time in a cheap 8-bit microprocessor. This is an excellent component for a simple cipher, though 8-bit substitutions alone are not very secure.

If we are willing to employ more of the address space of a microprocessor in a substitution table, then large substitution tables are possible. In an 8-bit machine with 16-bit addressing it is unlikely that more than 16 Kbytes could be given over to a substitution table. This puts the practical limit at 14 bits. If the table has to be loaded using the key, the time to do this is also important.

Building a 64-bit cipher (for example) out of 8-bit substitutions requires some way of mixing together the outputs of several substitutions. This illustrates the importance of permuting bits or characters, the so-called 'transposition cipher'.

Keyed substitution

A fixed substitution, for example one built into a read-only memory (ROM), can be a powerful component of a cipher, but a substitution that is unknown to the enemy because it is derived from the key can be even better. Fortunately it can be shown[7] that almost all 8-to-8 bit substitutions are acceptable ones, free of the flaws of linearity or insensitivity to certain input bits.

Often the requirement is for an invertible substitution or bijection, which means that the set of 256 byte values (for example) are a permutation of the set of numbers 0, 1, ... 255. Efficient methods of generating such permutations exist[8], but

economy of key bits is not a consideration if each key bit is being used for multiple purposes. Simpler methods are available.

For example, let $y = S(x)$ be the substitution to be generated, in which x and y take the values 0, 1, ... 255. Initially make $S(x) = x$. Then use the key, or part of the key, as a 'seed' to generate a pseudo-random integer sequence $R(i)$ for $i = 0, 1, \ldots 255$ where $0 \leq = R(i) \leq 256$. These random integers are used in turn to permute $S(i)$ with $S[R(i)]$. In this way each element of $S(x)$ is permuted with a randomly chosen other element of the array. If desired, the permutation process can be continued, to ensure a random result.

Transposition in modern ciphers

We have seen how substitution can be provided by table look-up, which is a fast operation either in a programmed microprocessor or purpose-built hardware. Transposition of bits in purpose-built hardware need be no more than a crossing over of connecting wires on a chip or board. In a serially operated device, it requires a number of delays and feedback paths and becomes complex.

Frequently the effect of transposition is provided by shift registers and by loading a set of 8-bit 'registers' in one sequence and reading them in another. The carry propagation in arithmetic operations can exploit the functions available in a processor to produce the effect of transposition, but the average length of a carry sequence is small, about log n bits for an n-bit number.

Variable or key-dependent transpositions of the classical form (bit or letter transposition) can be difficult to arrange in hardware. If all the permutations are to be available, a device like a miniature-scale telephone switch is required. In programmed form it is a little easier to do, but bit transpositions always tend to be inefficient and slow, hence the value of hybrid methods based on shifting, adding and XORing whole words.[9]

2.6. ATTACKS AGAINST ENCIPHERED DATA

Until the advent of digital computers the task of decipherment of cryptograms depended entirely on human skill; it was an esoteric art requiring considerable intuition on the part of the practitioner and a great deal of tedious work. Digital computers have changed this. For all the classical methods, analysis of enciphered text to identify the type of cipher can be carried out rapidly; once the cipher mechanism is known, the processing speed of computers allows quite crude methods to be used to break these classical ciphers.

Nevertheless, someone receiving a radio signal from outer space has a very considerable problem in interpretation — the modulation system of the signal may be complex, the 'plaintext' language is unknown, the message may be enciphered in an unknown cipher algorithm and the encipherment may be controlled by an unknown key. The message may not be intelligible, even with immense computing power and unlimited time.

The cryptanalyst trying to obtain the meaning of a message couched in an unknown cipher with an unknown key faces a task which is nearly as daunting. Unless a very elementary cipher has been used, identification of the encipherment algorithm itself can present a severe problem. When the cipher is known,

in most cases the key must be found before the message can be interpreted. Trying all possible values of the key to see if one of them fits is known as 'exhaustive search'. Finding the key by this method can be a monumental task when the number of possible values of the key is great.

Very often obtaining the key is a much more valuable prize than just the cracking of one message. It is likely that a whole stream of messages has been enciphered with this key and the whole stream becomes intelligible.

Classes of attack

In assessing the strength of a cipher system it is prudent to assume that the algorithm is known to the cryptanalyst and that the attempt to crack the enciphered message consists of trying to find the key. The cryptanalyst's task is most difficult when all he has to go on is ciphertext, with no knowledge whatsoever of the plaintext; this is the *ciphertext-only* attack. If there is no redundancy in the plaintext (for example, an arbitrary string of numbers) then it is impossible to find a key. For a given ciphertext, each key value gives for its plaintext an arbitrary string and all these strings are possible. Some known plaintext could resolve this problem, for example if the arbitrary message has a preamble in standard format. Keys can be tested until an instance of the preamble appears in the deciphered text; if the preamble is long enough then this may allow the key to be identified with confidence, since the preamble only appears for one key value. Otherwise it may be necessary to collect all possible candidate decipherments, thus identifying a set of possible keys. A knowledge of the representational code of a plaintext with, say, even parity, can give the cryptanalyst something to work on.

More favourable to the cryptanalyst is the situation where some plaintext and its matching ciphertext are available — the *known-plaintext* attack. The task is then to find a key corresponding to this match. If the length of the available text is sufficient, then the key can be identified with absolute confidence. Circumstances in which a plaintext–ciphertext pair is known are more common than one might imagine. For example, market changes may lead to predictable instructions being sent out to bankers or brokers. Sending enciphered press releases from a country to its embassy is another case in point. This is discouraged because of the working material it might afford a cryptanalyst seeking to break the diplomatic cipher. The remedy here is to ensure that the press releases are paraphrased before release.

The most favourable situation for the cryptanalyst arises from a *chosen-plaintext* attack. Here the cryptanalyst has somehow managed to introduce a plaintext of his own choosing into the encipherment process. The plaintext may be chosen to ease the task of key finding; for example, it may be helpful if the chosen plaintext could be 'all zeros'. The 'probable word' is an example of known plaintext, though its position in the text is unknown to the cryptanalyst. Sometimes the probable word is actually 'chosen plaintext'. An example of forcing a word into a cipher system occurred in the breaking of Enigma, where a bombing attack was made on a lightbuoy with the certainty that the comparatively rare word 'leuchttonne' would turn up in Enigma enciphered messages.[10]

In order to ensure the strength of a cipher it is best to make the most adverse

assumptions about the information available to the enemy. The least that is assumed in practice is that the cipher algorithm is known, together with enough corresponding plain and ciphertext to determine the key, in other words that a known plaintext attack is possible. For a more critical view of its strength, chosen plaintext should be assumed. A similar criterion, less discussed but worth considering, is that of chosen ciphertext. For example, could the enemy insert his own text on the line and somehow get access to the plaintext? A good modern cipher should resist all these attacks.

If the method of attack is to be by exhaustive key search using a computer, given a known plaintext–ciphertext pair, then we can estimate the mean time it will take to find a key based on assumed times to test each key and the number of possible different keys. We assume for the sake of illustration key testing times of 1 μs and 1 ms. These times approximate to the performance one might expect from special-purpose, large-scale integration (LSI) hardware and a microprocessor system, respectively. We assume key domains of 24, 32, 40, 48, 56 and 64 bits. In the left-hand part of Figure 2.20 we show the times taken to find correct keys, assuming that one key is tested at a time and that on average the correct key is found after working through half the key space. Any results above and to the right of the thick line represent conditions where the cipher is very weak. Results expressed in days may indicate conditions to be avoided for very important messages, whilst results expressed in years may be strong enough for most purposes. The results for times greater than 100 years are not shown, being beyond the normal expectation of human life.

Imagine, however, that a cryptanalyst has at his disposal a more powerful machine which is able to test many keys in parallel at the same time. Our imaginary machine contains many distinct units, each of which is programmed to explore a particular part of the key domain; eventually one of the units identifies the key and the machine signals success in its key search. For the sake of illustration we shall assume that such a machine can be built and that it is capable of testing 1 million keys simultaneously. This changes the situation completely. The right-hand part of Figure 2.20 shows the times now achieved in key sear-

Key size bits	*Single test*		*10^6 tests in parallel*	
	1 ms	1 μs	1 ms	1 μs
24	2.33 h	8.4 s	8.4 ms	8.4 vs
32	24.9 d	35.8 m	2.15 s	2.15 ms
40	17.4 y	6.4 d	9.2 m	550 ms
48	> 100 y	4.46 y	1.63 d	2.35 m
56		> 100 y	1.14 y	10.0 h
64			> 100 y	107 d

Fig. 2.20 Times for exhaustive key search

ching. With the powerful key search machine implied here, even a 64-bit key may not be large enough to give adequate security.

2.7. THE STREAM CIPHER

Where a communication system needs to handle characters in a stream, as, for example, when a simple terminal without storage is connected, this implies certain constraints upon the choice of cipher system that may be used. Each character must be accepted as it becomes available, enciphered and then transmitted. Clearly some form of substitution cipher could be used, but not a cipher which transposes characters.

Plaintext data arriving in this fashion is normally handled by what is known as a 'stream cipher'. A stream cipher may be synchronous, when encipherment depends on the current plaintext character and its position in the stream (in addition to the encipherment key) and the ciphertext is not influenced by the particular plaintext characters which have gone before. Enigma produces this kind of cipher. If the positional information becomes inaccurate, then the result may be completely unintelligible. Other types of stream ciphers are self-synchronizing; decipherment then depends on the current plaintext character and on a limited number of previous ciphertext characters; synchronism may be lost temporarily, but recovered as more text is processed. Because of the dependence of the current ciphertext on the immediately recent ciphertext, each ciphertext character depends on the current plaintext character, its position in the plaintext stream and also on the whole of the preceding plaintext stream.

The Vigenère cipher is an example of a synchronous stream cipher. The modulo 26 addition of a key text into a plaintext naturally takes place character by character and depends on relative position of current plaintext and key text; correct registration of plaintext and key text is essential. In that context we pointed out that a truly random stream of characters used as key material would yield an uncrackable ciphertext. The idea of adding a random stream into a plaintext arose as a result of an important invention by Vernam in 1917.

The Vernam cipher

At that time Vernam was working with other employees of the American Telephone and Telegraph Company on methods of encipherment for the printing telegraph. Vernam's important contribution was the bit-by-bit combination of random characters with characters of the plaintext, the combination being carried out by modulo-2 addition (the XOR function); the principle is illustrated in Figure 2.21. Random characters, chosen by picking numbers out of a hat, were punched on paper tape; this tape was fed through one reader in synchronism with the plaintext tape fed through another reader. A new tape was punched in a tape punch, enciphered in this neat and elegant operation. The encipherment key was the random number stream punched on the key tape, sometimes called the 'key stream'.

Decipherment was equally simple; all that was necessary was to take the received enciphered tape and feed this through a tape reader, combining it by bit-by-bit modulo-2 addition with a copy of the random key stream. In other words,

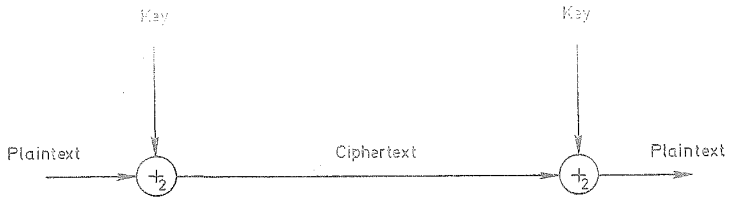


Fig. 2.21 The flow of data in the Vernam cipher

the process of decipherment used exactly the same mechanism and exactly the same key data as the process of encipherment. Figure 2.21 shows the data flows in the Vernam encipherment and decipherment. The modulo 2, or 'exclusive-OR', operation, used in the Vernam encipherment, has acquired great importance in designing modern cipher procedures. There is close similarity between Vernam's and Vigenère's methods except that one works with bits and the other with characters.

To operate Vernam's cipher it was necessary to provide sender and receiver with identical sequences of random numbers. Vernam's original idea was to use loops of tape, so that the random number stream was used over again as many times as necessary to encode a complete message. In order to generate longer number streams, two random number tape loops (with different random number streams) were used together, the tapes differing in length by one character. The two tape loops were run in synchronism and combined in an exclusive-OR operation, the result being combined with the plaintext. Thus, if one loop were 1000 characters long and the other 999, the number of combinations which would be produced before a repeat would be 999 000, but this is not a true indication of its complexity.

If tapes of truly random numbers can be provided at the two ends of a communication system using a Vernam cipher system, then the cipher is completely uncrackable. This is because, for any ciphertext with an unknown random key, all possible plaintexts of the same length are equally valid as decipherments. For very important communication channels it is worth going to the trouble of securely providing such random number tapes. Since the key is as long as the message, it is not generally useful, but there are situations needing great security in which the amount communicated is small but it is urgent and carrying bulk data as key is not difficult. This method, known as the 'one-time tape', has been used to protect channels such as the Washington–Moscow 'hot line'.

A similar method, operated by hand, is the 'one-time pad' which has been in the news more than once in connection with the unmasking of spies. Here the random number series is printed on tear-off pads, each sheet being used only once and then destroyed. Matching pads are held at sender and receiver locations.

As was pointed out in connection with the Vigenère cipher, so, with the Vernam cipher, it is important to avoid using the same key for more than one message, hence the use of the term 'one-time'. Use of the same key in encipherment for a number of plaintexts allows attacks either by a Kerckhoffs' superimposition or by elimination of the key by an exclusive-OR combination of the two ciphertexts.

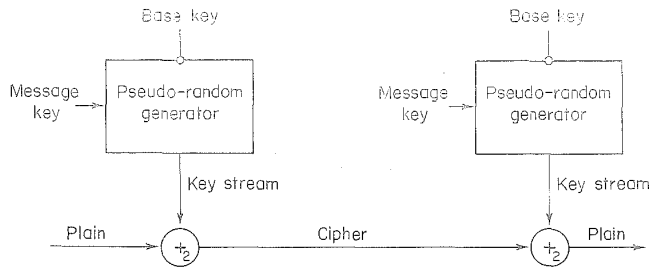


Fig. 2.22 Stream cipher with pseudo-random generators

Pseudo-random key streams

According to the Vernam principle the same random key stream is used for encipherment and decipherment. If a repetitive key stream is insecure and a completely random key stream is impractical, the compromise which is normally used is to employ a pseudo-random key stream generated by a finite state device at each end of the line as shown in Figure 2.22. Much of the ingenuity of cryptographers has gone into the development of these key stream generators. The key stream generator employs a secret key, the main cryptographic key, sometimes known in this context as the *base key*.

In order to synchronize the two ends of the line, the two pseudo-random key stream generators must begin with the same stored data, that is to say the same state of the machine. The number used to establish this state may be considered as part of the key, sometimes called the *message key* and it can be exchanged by a preamble to the main communication. Many systems allow the synchronizing data to be sent in clear through the network. Because synchronization can be lost due to a fault in the system or a bad communication line, practical systems which run automatically must be able to recover synchronism without human intervention. Effectively, this means that a portion of the channel capacity has to be given over to synchronizing information.

In some systems, the data being transmitted has considerable redundancy, perhaps because it is multiplexing a number of data streams. In this case, a loss of synchronism can be detected and both key stream generators stop their normal operation and enter into a resynchronizing sequence. In other systems, particularly those working at high speed or over networks with delay such as satellite links, the synchronizing information is sent continually and the two units resynchronize at a moment defined by a secret criterion based on the structure of the data being transmitted.

An example of this kind of key stream is given in chapter 4, section 4.4 — a method called output feedback.

2.8. THE BLOCK CIPHER

Block ciphers handle plaintext characters in blocks, operating on each block separately. The block size depends on the particular encipherment device in use;

the Data Encryption Standard (see chapter 3) is basically a block cipher operating on blocks of 64 binary bits.

An advantage of a block cipher is its ability in general to operate faster than a stream cipher. Data may be efficiently handled in blocks in applications such as file transfers. However, if there is no influence from other parts of a plaintext on the encipherment of each block this is a serious disadvantage. It means that if a block happens to occur more than once in a plaintext, then corresponding repeated ciphertext blocks will occur in the ciphertext, giving away information to an eavesdropper; this characteristic is very like that of material that has been encoded, as distinct from enciphered, when repeats may also be evident (indeed the simple block mode of using the Data Encryption Standard is usually known as 'electronic code book'). Even more seriously, an active line-tapper may be able to change the order of ciphertext blocks, delete selected blocks or duplicate particular blocks without risk of discovery.

Since the block cipher is effectively a substitution cipher acting on blocks instead of characters, if such a system is to be used with anything like adequate security, it is essential to use blocks of appreciable size in order to prevent a cryptanalysis based on frequency counts of block types.

Methods have been devised for using block ciphers as a basis for constructing stream ciphers. We shall see in chapter 4 how this has been done for the Data Encryption Standard.

2.9. MEASUREMENT OF CIPHER STRENGTH

Shannon's theory of secrecy systems

A most important contribution to the theory of encipherment systems was made by Claude Shannon, when he published his paper 'Communication theory of secrecy systems',[11] which followed soon after his better-known paper on the mathematical theory of communication. Shannon assumed that the cryptanalyst had only ciphertext to work on, which, from the point of view of cryptanalysis, is a more stringent assumption than the known or chosen plaintext assumption.

Shannon identified two significant classes of encipherment system — those that are unconditionally secure and those that are computationally secure. An *unconditionally secure* system is defined as one which defies cryptanalysis even though the cryptanalyst has unlimited computing power available to him. The only unconditionally secure cipher that is in common use is the one-time tape system, mentioned earlier, if used with a truly random key tape. As we have already seen all plaintexts with length equal to the ciphertext are decipherment candidates and there is no way of choosing between them, computational resources notwithstanding.

A particular cryptogram may be unconditionally secure if it is so short as not to contain sufficient material to lead to a unique solution. Even a monoalphabetic substitution ciphertext may fall into this class. Shannon defined the minimum length of text required to yield a unique solution as the 'unicity distance'. Stated simply, Shannon's criterion was that the redundancy in the plaintext must exceed the information contained in the key. For example, the key size for a monoalphabetic substitution cipher is 26! and $\log_2 26!$ is 88. If the redundancy

of English-language text is 50 per cent, then each character contributes about 3.3 bits of redundancy. Therefore any ciphertext with more than $88/3.3 \simeq 23$ characters can be used to determine the key. The unicity distance is 23 characters. Since Shannon's estimate of the redundancy of English included spaces, cryptanalysis of text without spaces requires a little more text. Still, the result is close to the lower limit observed in cryptanalysis by traditional 'craft' methods. For a one-time tape cipher, the unicity distance is infinite.

A ciphertext which contains sufficient information for cryptanalysis is said to be *computationally secure* if the task of solution is so large as to be impossible of execution with the largest conceivable computational power. The 'one-way' functions that we shall meet in chapter 8 are of the class that are easy to compute in one 'direction', but are required to be computationally infeasible in the other.

There is no accepted definition of what is computationally infeasible, bearing in mind that extremes of parallel processing can be used, that time-memory trade-offs are possible and that some notion of the acceptable time for a solution must be obtained. Furthermore, processing and storage technology continually improve. It would generally be agreed that a calculation employing more than 10^{25} steps is not feasible today. Perhaps 'step' should be defined as 'that which an LSI chip can do in 1 μs'. To get a more absolute limit, going beyond our present technology, it is possible to set a thermodynamic limit.

Limits of computation

A reasonable assumption is that each logical step consumes energy kT, where k is Boltzmann's constant and T the absolute temperature. From a calculation of the amount of heat falling on the earth from the sun and assuming that the cryptanalytic computation must take place at a temperature of 100 K, it can be shown[12] that the number of elementary operations obtainable in 1000 years is about 3×10^{48}. The other significant consideration in a time-memory trade-off is the amount of available store for the computation. Let us assume that somehow a store can be constructed in which each bit requires 10 atoms of silicon; this store shall be 1 km high and cover all the land mass of the earth (or shall be placed in orbit as a new moon of equivalent mass). The number of bits that can be stored is about 10^{45}.

Being cautious in placing our limits of computational infeasibility, let us define these as 10^{50} operations in 1000 years and a store of 10^{50} bits. Any computations requiring more than this amount of computing power can be said to be totally impossible in the time scale stated. In practical terms, computations requiring very much less power than this are also likely to be impossible. A feel for the scale of the problem can be gained by looking at cryptanalysis of the Lucifer system[13] with its key of 128 bits; if this can only be attacked by exhaustive search for the key, given a known plaintext–ciphertext pair, then the number of keys that must be tested is about 3×10^{38}. If one Lucifer key can be tested every picosecond, then the time taken to find a solution is about 10^{19} years.

Key size and its effect on the time for exhaustive search set an upper limit on the strength of a cipher, but it is a great mistake to judge the strength of a cipher in this way. The history of ciphers is full of 'unbreakable ciphers' with large key spaces. Key search is no more than the universal method on which to fall back

when all else fails. The strength of a cipher is a negative quality and depends on everyone's inability to find a feasible way of breaking it. It can never be guaranteed except for an unconditionally secure cipher. In chapter 8 we shall describe how 'complexity theory' might one day produce proof of the difficulty of breaking ciphers. At present this is just an aspiration. The best that can be done is to show that certain ciphers are as difficult to break as the difficulty of certain computational problems that mathematicians generally agree to be hard. The possibility of improved computational methods which make breaking feasible cannot be ruled out.

Looked at from the point of view of the theory of computational complexity, some systems can be proved to be unconditionally secure, but computational infeasibility of any cryptanalytic problem cannot be demonstrated. For this reason it is usual, when assessing the strength of a cipher, to postulate an attack under conditions which favour the cryptanalyst. This departs from Shannon's assumption of a ciphertext-only attack and makes the assumption of a known-ciphertext or chosen-ciphertext attack. It is on such assumptions that the strength of a modern cipher system is judged.

An application of Shannon's theory

An interesting application of Shannon's theory is to the solution of messages formed by adding two English texts. The operation can be modulo 2 addition of bits (Vernam) or modulo 26 addition of a letter code (Vigenère) — it makes little difference. From one point of view, each text is enciphering the other. Each text has a redundancy of about 80 per cent. The amount of key needed to solve it is provided by the other's redundancy. In other words, the 20 per cent equivocation in each text still leaves enough redundancy (60 per cent) to solve the problem. This, even if it were an exact argument, would not show that it could be done in practice because our knowledge of the structure of English is not perfect and long messages would be needed, with correspondingly long computation, to exploit all the redundancy which Shannon has estimated. In practice, by guessing first at one message and then the other the problem can often be solved. For a starting point, a probable word or phrase in one or other text is needed.

The same method of solution can be applied to an equivalent problem — solution where two texts have been additively enciphered with the same key stream. Such a solution gives both the texts and a sample of keystream for further analysis.

2.10. THREATS AGAINST A SECURE SYSTEM

Threats against any protected system, be it a communications system or a data storage system, can be classified conveniently into two categories — the passive and the active threat. A passive attack against a system is merely an attempt to get round the protective barriers and read what is being transferred or stored, but without attempting to alter it. This type of attack has been perpetrated against communication systems ever since the invention of the electric telegraph. In the days before multiplexing it was very simple indeed to introduce a wire tap onto a transmission line (Figure 2.23). Telephone tapping of a local connection is all too easy to achieve. The various forms of multiplexing now in operation and the

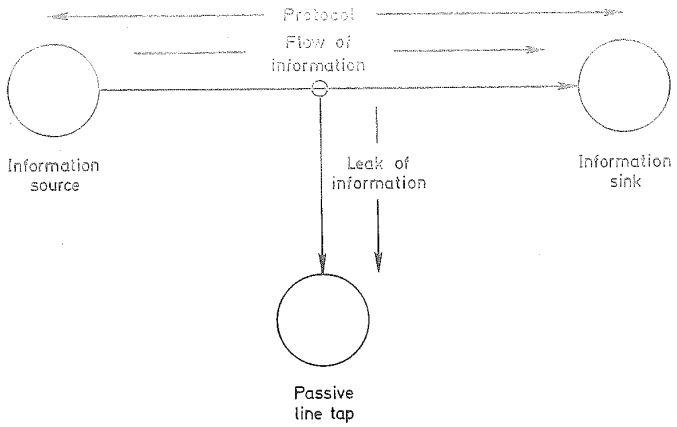


Fig. 2.23 A passive line tap

complex protocols in data networks demand greater sophistication of the intruding line-tapper, but this is well within the compass of the technologist criminal. It is often thought that multiplexing provides in itself a kind of safeguard against the intruder, but this is a dangerous and fallacious belief. It is well known that in the USA a foreign power made a practice of listening-in to microwave links (no physical line tap then being necessary) and selecting interesting telephone calls by recognition of the in-band dialling tones used to set up the calls.

A passive line tap need not break into the protocol controlling the flow of data. This simplifies the task of the passive line tapper enormously.

Detection of a passive line tap is feasible where a physical communication link is involved, based on precise measurements of line characteristics, but clearly this is impossible where a microwave link is tapped. Defence against passive line-tapping intrusion must therefore be based on protecting by encryption the data transmitted; it cannot hope to succeed by protecting the transmission line itself, except where the line is totally protected physically, an unusual state of affairs.

Active line taps

The active threat is potentially far more serious. Here the intruder seeks to alter the stored or transmitted data, and will hope to do this without discovery by the legitimate data owner or user. Alteration of stored data may take place either through misuse of the normal access channels or by re-writing data on exchangeable storage media. We are not so much concerned here with the access rules, though this is an important subject, but rather with the techniques for protecting data against attack via clandestine means. Use of encryption can protect against undetected alteration of the data by arranging that the encrypted data is structured in such a way that meaningful alteration cannot take place without cryptanalysis; we shall discuss this at greater length when we consider secure data formats using encryption. Where an intruder has illegal access to data-recording media, deletion of data cannot be prevented. Clearly this is a very

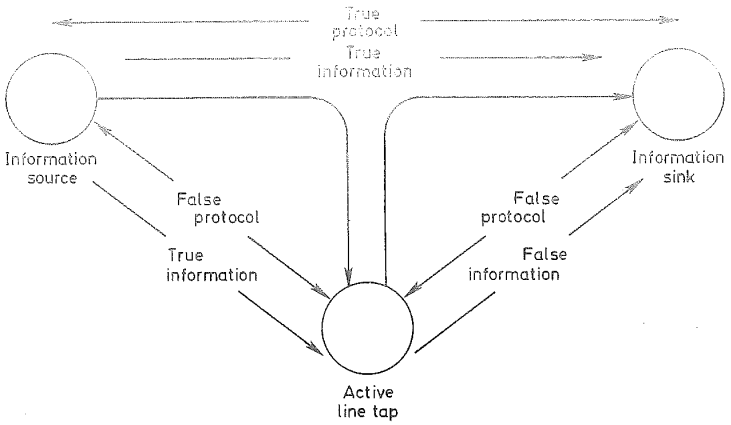


Fig. 2.24 An active line tap

serious matter and demands the creation of a secure physical environment for storage of tapes and discs, together with back-up copies at a different storage location.

An active line tap on a communication link is almost as difficult to prevent as is a passive line tap except in radio channels. Detection is much easier and may even be possible without encipherment or similar logical schemes. Successful alteration of data requires time to carry out and precise measurements of transmission times may indicate the presence of intrusion. Whereas a microwave link was easily tapped in passive mode, this type of link would present a severe challenge to an active line tap. If a physical communication link is under constant surveillance by its owner, with traffic constantly flowing and being monitored, then it may be almost impossible for the intruder to introduce his tapping connections without detection. The type of configuration that a line tapper would require is shown in Figure 2.24.

To operate an active line tap successfully is not a simple task. In Figure 2.24 communication between source and sink is controlled by a protocol, called in the figure 'true protocol'. The line tapper must interrupt this protocol and conduct a separate 'false protocol' with source and with sink, thereby deceiving each that they are still in direct communication. The true information flows from the source to the active line tap; false information flows from the tap to the information sink. The task of the line tapper is probably simplest when source and sink are allowed first to establish a genuine communication channel, to be taken over later by the line tapper. With some added complication the active tap takes part in the process of setting up the channel.

Methods of protection

Protection of transmitted data against an active line tap is based on the same principles as those mentioned earlier in connection with the protection of stored data. It is essential to prevent undetected alteration, addition, deletion and replay. In

any one block of transmitted data the first three risks can be averted by arranging the encrypted data structure in a chain, so that any portion of the encrypted data is dependent on the corresponding plaintext data and also on the preceding plaintext; meaningful alteration cannot be carried out without finding the encryption key by cryptanalysis. We shall discuss chaining techniques in some detail in chapter 4 when we come to consider the modes of operation used with the US Data Encryption Standard.

Replay is the attack in which the intruder records an enciphered transmission and later sends it again from his line-tapping transmitter. For example, the aim might be to make money transfers to a bank account appear to take place more than once. Replay cannot be prevented, but it can be detected by including in the plaintext sufficient block identification data, a block number (either unique or cyclic with a large range) and, possibly, a time-stamp. The receiver would ignore any enciphered message whose decrypted plaintext did not contain valid identification. Detection of unauthorized changes is part of the subject called *authentication* and is treated in chapter 5.

2.11. THE ENCIPHERMENT KEY

In all but the most elementary of the systems we have discussed in this chapter, the role of the encipherment key is most significant; key-less encipherment algorithms, which always behave in a predictable manner, are only of academic interest; the security of such systems depends on keeping the design details secret and when the secret is lost a new design is needed to replace the lost cipher.

In keyed algorithms the encipherment process is carried out under the control of the key, which influences the behaviour of the encipherment algorithm. This property applies equally to substitution, transposition and product ciphers; it is particularly important when we come to machine ciphers. In the latter, capture of a machine may expose the detailed construction of a secret encipherment algorithm to an opponent. However, if the opponent does not know the relevant encipherment key, then knowledge of the algorithm alone should be totally insufficient to permit cryptanalysis of an enciphered message. It is a sound principle, much followed in modern practice, that security of a cipher system should not depend on the secrecy of the algorithm, but on the secrecy of the encipherment keys.

This principle places a considerable responsibility on the designer of the system under which keys are generated, distributed, used and destroyed after use. In the past the key material has been sent to encipherment users by human courier, but this is too slow and inconvenient for many of today's applications. Therefore, means must be devised for transporting keys more rapidly and efficiently. Often this implies sending the keys via the communication channel which is itself the subject of encipherment protection. Here we need an hierarchy of keys which distinguishes those keys which are used for data protection (and which may therefore be much in use) from those used solely for the protection of keys in transit. Keys used for encipherment of keys are used far less frequently and may be transported by less efficient means, such as a physical visit; the very fact that they are used less frequently means that they are less exposed to cryptanalytic attack.

It is important that the system designer gets the key management system right and does not leave loopholes exploitable by an intruder. We treat this subject in detail in chapter 6.

REFERENCES

1. Kahn, D. *The Codebreakers*, Macmillan, New York, 1967.
2. Gaines, H.F. *Cryptanalysis*, Dover, New York, 1956.
3. Friedman, W.F. *The Index of Coincidence and its Applications in Cryptography*, The Riverbank Publications, Aegean Park Press, Laguna, 1979.
4. Hinsley, F.H., Thomas, E.E., Ransom, C.F.G. and Knight, R.C. *British Intelligence in the Second World War, Vols I and II*, HMSO, London, 1979 and 1981.
5. Beker, H. and Piper, F. *Cipher Systems: The Protection of Communications*, Northwood Books, London, 1982.
6. Davies, D.W. 'The Siemens and Halske T52e cipher machine', *Cryptologia*, 6, 4, 289, October 1982; 7, 3, 235, July 1983.
7. Gordon, J.A. and Retkin, H. 'Are big S-boxes best?', *Cryptography* (Proceedings, Burg Feuerstein, 1982, ed. T. Beth), Lecture Notes in Computer Science 149, Springer Verlag, Berlin, 1983.
8. Ayoub, F. *Some aspects of the design of secure encryption algorithms*, Hatfield Polytechnic, School of Engineering, Ph.D. Thesis, March 1983.
9. Brüer, J-O. 'On non-linear combinations of linear shift register sequences', *Proceedings, IEEE 1982 International Symposium on Information Theory, Les Arcs, France. Also* appeared as Internal Report LiTH-ISY-I-577, Linkoping University, March 1983.
10. Johnson, B. *The Secret War*, BBC Publications, London, 1978.
11. Shannon, C.E. 'Communication theory of secrecy systems', *Bell System Technical Journal*, **28**, 656, October 1949.
12. Davies, D.W. *Limits of computation*, NPL Internal Note, 1978.
13. Feistel, H. 'Cryptography and computer privacy', *Scientific American*, **228**, 5, 15, May 1973.

Chapter 3 THE DATA ENCRYPTION STANDARD

3.1. HISTORY OF THE DES

Why do we need a data encryption standard, since proprietary encipherment systems are available from many manufacturers? Users wishing to protect communication on a line need only purchase a pair of matching encipherment devices, install them, enter matching keys and they are ready to communicate securely. Whenever communication is internal to a particular group, protection of transmission by encipherment presents little in the way of organizational problems, such as reaching agreement on a particular method of encipherment. Until recently most data communication has been within single organizations, often carried on a private network of leased lines. In these conditions a standard is not needed.

When protected communication is required between users who belong to different organizations, this may not be so easy to arrange. The organization may have chosen encipherment devices from different manufacturers, in which case the devices are likely to be totally incompatible and unable to communicate either securely or otherwise. Furthermore encipherment must often be introduced in the context of standard communication protocols; this may place new restrictions on the means of encipherment. This kind of interworking is becoming increasingly common. Therefore, there is a case for considering standard encipherment algorithms for widespread use. Of necessity any standard encipherment algorithm must be public knowledge and this is a departure from previous cryptographic practice where users attempted to keep the nature of the encipherment algorithms in use a secret. The entire security of a system based on a public standard must rest on the strength of the algorithm and the secrecy of the encipherment key.

The initial move towards a public encipherment standard came from the US Federal Government, which, if it chose, could have dictated to its agencies involved in data processing what existing proprietary algorithm they should use. Encipherment had, of course, been in use by the US Government before this initiative; such encipherment was designed to protect data covered by the US National Security Act of 1947 or the Atomic Energy Act of 1954 and similar legislation. The new initiative was aimed at areas where the responsible authority deemed that the data in question, though not covered by legislation demanding encipherment (that is to say, not 'classified'), yet merited cryptographic protection; in general the application was, therefore, outside the military, diplomatic and other areas of national security.

The role of NBS

The US Federal department given the task of initiating this development was the Department of Commerce; within this department the responsible agency was the National Bureau of Standards (NBS). (It is now renamed the National Institute for Standards and Technology (NIST).) Under the Brooks Act (Public Law 89-306) NBS had the responsibility for setting Federal Standards for the effective and efficient use of computer systems; the relevant clause of PL 89-306 reads: 'The Secretary of Commerce is authorized ... (2) to make appropriate recommendations to the President relating to the establishment of uniform Federal automated data processing standards.' The standards which have been produced as a result of this legislation are expressed in a substantial series of publications under the general title of Federal Information Processing Standards; they cover hardware, software, data, ADP operations, etc.

Under this general remit NBS initiated a study programme on computer security. One of the early actions in this programme was to express the need for a standard for data encryption. The Bureau took the view[1] that purchase of encipherment devices from commercial suppliers must be subject to the principle of 'caveat emptor' or 'buyer beware'. An NBS statement averred that 'the intricacies of relating key variations and working principles to the real strength of the encryption/decryption equipment were, and are, virtually unknown to almost all buyers, and informed decisions as to the right type of on-line, off-line, key generation, etc., which will meet buyers' security needs have been most difficult to make'.

The NBS's responsibility extended primarily to the protection of data as processed and transmitted by Federal agencies; however an important secondary area of responsibility as part of the Department of Commerce was to the general buyer of encipherment equipment. An interesting sidelight is thrown by the NBS's view that much of the proprietary (i.e. non-standard) encipherment equipment available came from outside the USA. Therefore the preparation of a standard for data encipherment would not damage the trade of US manufacturers to a serious degree.

In 1974 the US Congress passed a Privacy Act, which, strangely, did not address directly the question of data security. However, data security is an important tool in attaining in a satisfactory manner the requirements of legislation of this type, and the Office of Management and Budget assigned to NBS the responsibility of developing computer and data standards to meet the needs of this Act.

The NBS study programme got under way effectively in 1972. A search was begun for an appropriate encipherment algorithm which could form the basis of a Federal Information Processing Standard. The requirements for this were as follows:[2]

1. It must provide a high level of security.
2. It must be completely specified and easy to understand.
3. The security provided by the algorithm must not be based on the secrecy of the algorithm.
4. It must be available to all users and suppliers.
5. It must be adequate for use in diverse applications.
6. It must be economical to implement in electronic devices and be efficient to use.

7. It must be amenable to validation.
8. It must be exportable.

'Economical implementation' virtually demands that the algorithm be realizable in special purpose large-scale integration (LSI) hardware; mainframe computers are not at their best in carrying out the bit manipulations that form typical components of encipherment algorithms. Acting under the remit expressed in the eight points listed, the NBS team was not looking for ideas needing a lot of further development, only methods which were already well developed were sought.

In May 1973, NBS issued a call for ideas fitting these needs. The response was disappointing; many mathematicians sent them outlines of algorithms which needed development. One suggestion received was virtually a restatement of the 'one-time tape' method, based on the Vernam invention. None of the offerings came anywhere near filling the stated requirements. In August 1974 a second call for ideas was issued. In response, several algorithms were sent in; some of these were too specialized, some were not strong enough. However, one algorithm submitted showed great promise in meeting the requirements. This submission came from International Business Machines (IBM).

The IBM Lucifer cipher

The IBM submission to NBS was a development from an encipherment scheme invented in the early 1970s; this was the IBM 'Lucifer' encipherment algorithm, a 'product' cipher, one of the early applications of which is understood to have been in a bank automatic-teller terminal. As the eventual Data Encryption Standard (DES) was more or less directly developed from the Lucifer design it is worth examining Lucifer in a little detail. Lucifer certainly met the NBS requirement that the algorithm should be made fully public; it had been the subject of several articles quite freely available in various journals and reports.[3,4]

In chapter 2 we met various substitution and transposition ciphers and we saw that, used alone, many of these are very weak. However, as was pointed out by Shannon,[5] cipher operations which are weak in themselves can be combined together to form something much stronger; this is the concept of the product cipher. A product cipher may take a block of plaintext and transpose the order of its binary bits (called 'diffusion' by Shannon), following this with a systematic substitution of bit patterns with others according to look-up or substitution tables (called 'confusion' by Shannon). These two steps may be repeated a number of times. Figure 3.1 illustrates the process.

In Figure 3.1 the first logical step is a substitution, in which the bits of the plaintext block are taken in groups and replaced by other bit patterns of the same size according to substitution tables (S-boxes), with no expansion of data. The second step is a transposition, or re-ordering, of the bits of the whole output from the first step. The third step is a substitution, similar to the first step. This is followed by a further transposition and so forth. Thus transposition and substitution alternate until the ciphertext produced as a result of the total process is unpredictable from the plaintext without knowledge of the intervening transformations (the actual number of steps employed is set by the designer of the

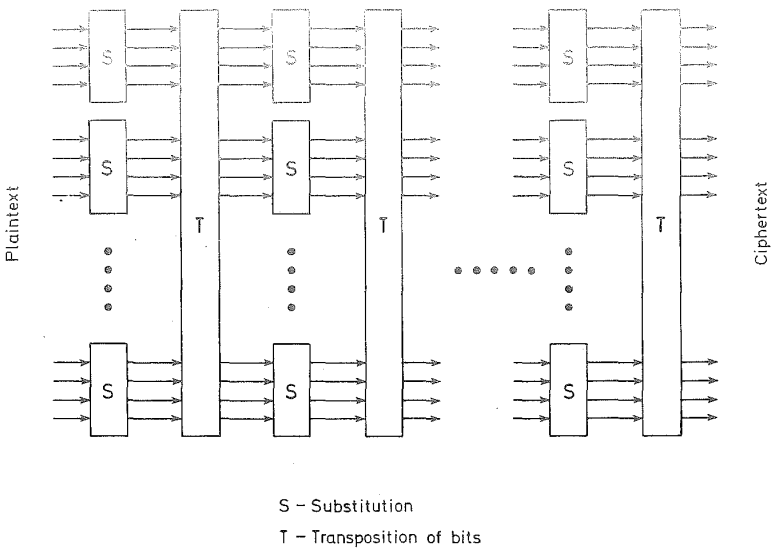


Fig. 3.1 A product cipher with transposition and substitution

encipherment). Decipherment by the possessor of this knowledge is carried out by running the ciphertext data back through the process, using the inverse of each S-box and transposition; without this knowledge a plaintext block should not be deducible from the ciphertext. Greatest strength of encipherment is achieved if the S-boxes used at each stage are different. If we say that the S-boxes are to carry out transformations on groups of 4 bits, then the total number of different S-boxes that can be specified is $(2^4)!$ or about 2×10^{13}, so there appears to be plenty of choice, although few of the choices are actually suitable.

The description of the transformations can be expressed in terms of a key, which should be shared knowledge between communicating parties and must, therefore, be readily transportable. The size of this key can be calculated precisely on certain assumptions. We have already postulated 4-bit S-boxes — now suppose that the transposition should be the same at all steps, and, therefore, need not be specified in the key. Take a plaintext block of 128 bits, the Lucifer block size, then at every stage thirty-two S-boxes will be required. An S-box usually comprises a read-only-memory (ROM) and in this case it stores sixteen words of 4 bits, but the information required to specify an *invertible* transformation of 4 bits is $\log_2(16!) = 44.2$ approximately. If there are 16 stages of transposition/substitution, then the total size of the specification of the transformations, i.e. the key size, will be $32 \times 45 \times 16$ ($= 23\,040$) bits. This is clearly an inconveniently large key.

In the Lucifer cipher only two different S-boxes were specified; the selection of the S-boxes to be used at each substitution step was based on the digits of the encipherment key; the S-box specifications themselves were part of the algorithm and not affected by the encipherment key.

For a 16-cycle set of transformations the key size would need to be 32×16 (=512) bits. In Lucifer the key size was further cut down to 128 bits by using each key bit four times. The key was also involved in a process called 'interruption', giving a modulo-2 combination of selected key bits with the results of the substitution and transposition operations; the cycle followed the order, confusion, interruption, diffusion. The details of the Lucifer operation were public knowledge, as were the specifications of the transpositions and S-boxes. The whole of Lucifer's strength depended on the complex series of substitutions, with interposed transpositions, and on the secrecy of the encipherment key.

The Lucifer block size was 128 bits, with no data expansion in the encipherment process, and the key size was also 128 bits. Thus the number of different mappings of plaintext onto ciphertext under control of the encipherment key was 2^{128}, compared with the total number of ways of mapping 128 bits onto a 128-bit field with is $(2^{128})!$, a very much larger number.

The process of establishing the DES

The IBM reply to the second NBS solicitation for candidate encipherment standards was based on an algorithm of this type. During the months that followed the submission of the algorithm the proposal underwent government scrutiny to determine its suitability as a Federal standard. The National Security Agency (NSA) was called in by NBS to carry out an exhaustive technical analysis of the algorithm. During this time NBS and IBM negotiated the terms under which this piece of IBM intellectual property might be made available to others for manufacture, implementation and use. IBM agreed to grant non-exclusive, royalty-free licences for the manufacture or sale in the USA of DES devices where otherwise this would infringe IBM patents. The agreement was eventually recorded in the 13 May 1975 and 31 August 1976 issues of the Official Gazette of the United States Patent and Trademark Office. In March 1975 the description of the algorithm was issued for general public comment and in August 1975 a draft of the proposed standard was published.

The comments received showed a wide range of views. Many comments concerned the ways in which encipherment should be implemented in various computer architectures. The recommended modes of use of the standard algorithm are examined in chapter 4. Others were directed at the cryptographic strength of the algorithm. We shall return to this subject in the present chapter.

The process of comment, discussion and reply occupied about 18 months, following which the data encipherment algorithm was adopted as a Federal standard on 23 November 1976. It was published as FIPS Publication 46, entitled *Data Encryption Standard*, on 15 January 1977,[6] becoming mandatory 6 months later on 15 July 1977. After that date Federal agencies were expected to comply with its provisions, or show an acceptable reason for not doing so.

The abbreviation DES has achieved very wide currency for the Data Encryption Standard. The algorithm has also been adopted by the American National Standards Institute, where it is known as the Data Encryption Algorithm (DEA). We will now describe the way in which the algorithm carries out the encipherment.

3.2. THE ALGORITHM OF THE DATA ENCRYPTION STANDARD

A logical flow diagram of the DES is shown in Figure 3.2 which encompasses both encipherment and decipherment. The operation has two 64-bit inputs, the plaintext (or ciphertext) block and the encipherment key block, and one 64-bit output, the ciphertext (or plaintext) block. The algorithm transforms plaintext into ciphertext or vice versa, depending on the mode in which it operates. Of the 64 bits in the encipherment key block, only 56 enter directly into the algorithm. In the Federal standard the eight remaining bits take the values for odd parity in each 8-bit byte of the key block.

The building blocks of the algorithm are permutations, substitutions and modulo 2 additions (XOR). Permutations in the DES are of three kinds, straight permutations, an expanded permutation and permuted choices. Figures 3.3 (a), (b) and (c) illustrate the nature of these operations (the permutations actually used are defined in later figures). In a straight permutation bits are simply reordered. In the expanded permutation some bits are duplicated and the whole resulting field reordered. In permuted choices some bits are ignored and the remainder reordered.

Substitutions in the DES are known as S-boxes and are specified by eight different tables. In Lucifer the input and output of the S-boxes were each of 4-bits. In the DES the S-boxes have 6-bit inputs and 4-bit outputs.

The first operation to which the input data is subjected is the Initial Permutation, which has a highly regular character. The effect of this may be seen as an acceptance of data, an octet at a time, in eight shift registers (one bit to each register). As the successive input octets are accepted, their bits are distributed

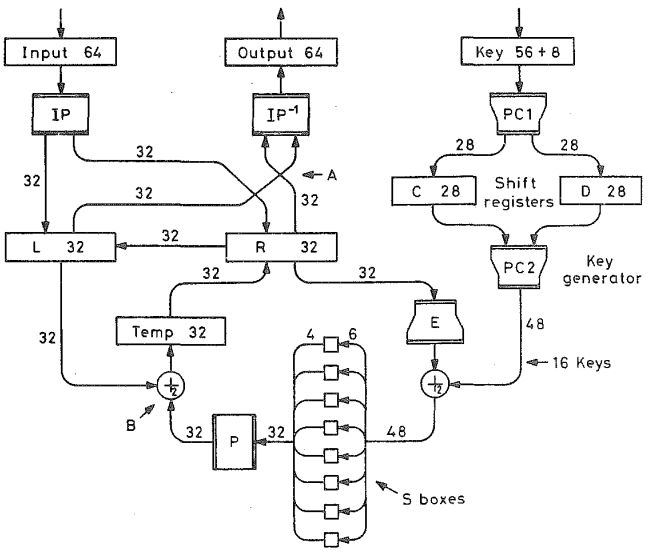


Fig. 3.2 The logical structure of the DES

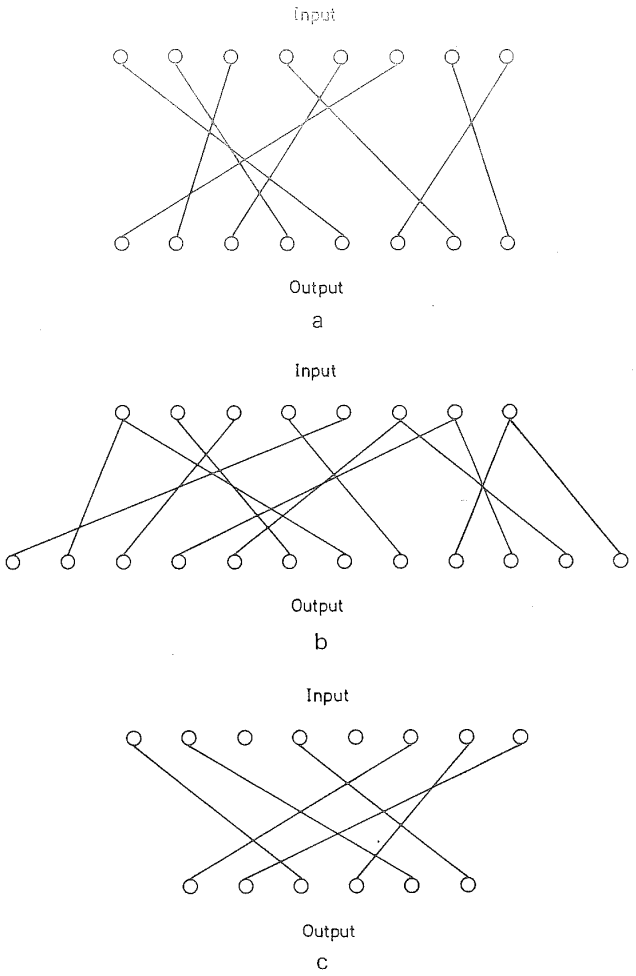


Fig. 3.3 Three types of permutation, a a straight permutation, b an expanded permutation, c a permuted choice

to the eight shift registers. When the input is complete and all 64 bits accepted, the contents of the shift registers are placed in sequence. The exact way in which the bits are permuted is shown in Figure 3.4. In this permutation input bit 1 emerges as output bit 40, input bit 2 emerges as output bit 8, etc.; output bit 1 corresponds to input bit 58, output bit 2 corresponds to input bit 50, etc.

One result of this permutation is that for a code such as American Standard Code for Information Interchange (ASCII), with every eighth bit representing parity, all the eight parity bits of the input block will be collected together into one octet at the beginning of the block, before transposition and substitution begins. There is, however, no evidence that this initial regularity introduces any

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

Fig. 3.4 The Initial Permutation of the DES

weakness into the encipherment operations; this is because of the powerful diffusion properties of the algorithm. It is understood that the algorithm includes this permutation for reasons of convenience in implementation; it has no cryptographic value.

Before the permuted and substituted data is available for reading from the output register at the end of the complete encipherment operation, it is subjected to the exact inverse of the Initial Permutation.

After the Initial Permutation, the 64-bit data field is split into two 32-bit fields which are sent to registers L and R, part of the main loop of the algorithm logic. Then the action of the main loop of the algorithm begins.

From register R the 32 bits are passed to the Expansion Permutation E, which is a regular transformation in which half the bits are replicated, so that the output of the permutation consists of 48 bits. For each of the four octets the first, fourth, fifth and eighth bits are replicated. The full permutation is expressed in Figure 3.5. In this permutation output bit 1 corresponds to input bit 32, output bit 2 corresponds to input bit 1, etc. The other output from register R is a direct 32-bit transfer to register L. We shall see later what happens to the displaced contents of register L.

The 48-bit output from the Expansion Permutation is next passed to an exclusive-OR operation in which it is combined with a selection of bits from the encipherment key. The 48-bit output from the exclusive-OR operation is split into eight sextets for input to the set of eight S-boxes, thus each S-box receives a 6-bit input, but produces only 4 bits as output. The individual S-box look-up tables are as shown in Figure 3.6. The layout of these tables requires further explanation. The

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

Fig. 3.5 The Expansion Permutation (E) of the DES

		Column															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S1	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
Row	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
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	12
S5		2	12	4	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	12	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

Fig. 3.6 The S-box tables for the DES

input to each consists of 16 bits, of which the first and the last are taken together as a 2-bit number which has the effect of selecting the row in the S-box table; bits 2 to 5 of the input to each S-box, together giving values in the range 0 to 15, select the table element in the appropriate row (the left bit is most significant); each row of an S-box table represents a bijection from bits 2 to 5 of the S-box input. We shall return to the question of the structure of the S-boxes.

The 4-bit outputs from all the S-boxes are collected together as a 32-bit field for input to the permutation P, which is a rather irregular permutation, quite unlike the regular structures of IP and E. The details of the permutation P are as shown in Figure 3.7; this permutation neither replicates nor removes bits and therefore

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

Fig. 3.7 Permutation P of the DES

has a 32-bit output. In this permutation bit 1 of the output corresponds to bit 16 of the input, bit 2 of the output corresponds to bit 7 of the input, etc.

To complete the main loop of the algorithm the output from permutation P is combined with the erstwhile contents of register L in an exclusive-OR operation and the result moved into register R. In order to achieve this operation without a clash on register allocation, a temporary register TEMP is shown in Figure 3.2 between the exclusive-OR and register R.

The cycle of the main loop we have described here is repeated sixteen times before the contents of registers R and L are assembled in a 64-bit block *in the sequence R, L,* then subjected to the inverse of the Initial Permutation already mentioned and transferred to the output register. Note the final swap of registers R and L before the inverse Initial Permutation. The reason will appear shortly. Each cycle of the main loop is usually known as a 'round'.

It now remains to describe how the encipherment key participates in the process. We mentioned that the 48-bit output from the Expansion Permutation E is combined with selected bits from the encipherment key before input to the S-boxes. The selected bits are different for each of the sixteen rounds. At the right-hand side of Figure 3.2 the method of generating these bits is shown. After input of the key to the key register, the key is read into the Permuted Choice 1 (PC1) where every eighth bit is eliminated, otherwise the structure of this permuted choice is very similar to that of the Initial Permutation, probably signifying a similar kind of implementation; the structure of PC1 is given in Figure 3.8. Here bit 1 of the input to register C corresponds to bit 57 of the encipherment key and bit

C	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
D	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

Fig. 3.8 Permuted Choice 1 of the DES

1 of the input to register D corresponds to bit 63 of the encipherment key. The 56-bit field produced as the output from this operation is split into two 28-bit fields and loaded into registers C and D. The latter are cyclic shift registers; for encipherment they are cycled left by either one or two bit positions for each of the sixteen rounds of the algorithm. The schedule of left shifts for encipherment is given in Figure 3.9.

To produce the selection of 48 bits required for combination with the E output in the main loop of the algorithm, the combined contents of C and D are sent to a further Permuted Choice (PC2). This has a rather irregular structure, expressed in Figure 3.10. Bit 1 of the output corresponds to input bit 14 (of the combined C and D), output 2 corresponds to input 17, etc. The output from PC2 consists of 48 bits. For each round of the algorithm the output of PC2 forms the 'sub-key' used in the main loop.

Cycle	*K block*	*Number of left shifts before PC2*
1	K_1	1
2	K_2	1
3	K_3	2
4	K_4	2
5	K_5	2
6	K_6	2
7	K_7	2
8	K_8	2
9	K_9	1
10	K_{10}	2
11	K_{11}	2
12	K_{12}	2
13	K_{13}	2
14	K_{14}	2
15	K_{15}	2
16	K_{16}	1

Fig. 3.9 Schedule of left shifts for encipherment

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

Fig. 3.10 Permuted Choice 2 of the DES

Cycle	K block	Number of right shifts before PC2
1	K_1	0
2	K_2	1
3	K_3	2
4	K_4	2
5	K_5	2
6	K_6	2
7	K_7	2
8	K_8	2
9	K_9	1
10	K_{10}	2
11	K_{11}	2
12	K_{12}	2
13	K_{13}	2
14	K_{14}	2
15	K_{15}	2
16	K_{16}	1

Fig. 3.11 Schedule of right shifts for decipherment

This description of the structure of the encipherment algorithm is now complete. Decipherment is carried out very conveniently by a similar process, the only difference being in the generation of the sub-keys. For decipherment the cyclic shifting registers C and D are cycled to the right, the opposite sense from that used in encipherment. The schedule of right shifts for decryption is given in Figure 3.11. Note that in encipherment a shift takes place before the first sub-key emerges but not in decipherment. The result is that in decipherment the same set of sixteen sub-keys is produced, but in the opposite sequence.

The ladder diagram

The method of operation of the algorithm can be understood more easily if we consider the ladder diagram shown in Figure 3.12. This illustrates the relationship between the successive rounds of the algorithm. We see how the exchanges between registers R and L are made under the influence of the sub-keys, k_1 to k_{16}; the latter are the 48-bit patterns produced by operating on the main encipherment key; the action of the permutation E, the S-boxes and the permutation P are subsumed in the function F in this diagram. For the sake of economy of space only three of the sixteen rounds are shown.

The ladder diagram also enables us to illustrate quite neatly the relationship between encipherment and decipherment. In Figure 3.13 we show the process of encipherment followed immediately by the process of decipherment. The ladder has been 'untwisted' to make its nature more clear. The symbols R and L refer to the contents of the respective registers at each stage in the process. Note that the sub-keys are presented for decipherment in the opposite order from that in which they were used for encipherment. At each point in the ladder for

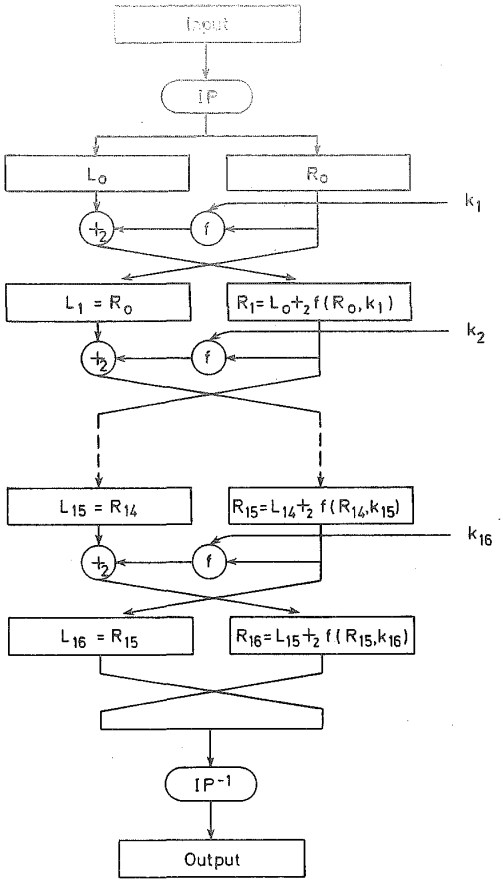


Fig. 3.12 Ladder diagram representation of the DES .

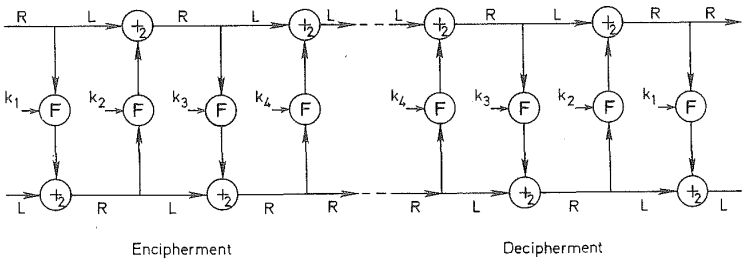


Fig. 3.13 Encipherment and decipherment mirrored in a ladder diagram

encipherment it is possible to establish that the state of the data is exactly the same as that of the corresponding state in the ladder for decipherment. The structure of the encipherment operation is exactly mirrored by the decipherment operation at their mutual boundary.

An algebraic representation

This result may also be expressed in algebraic form. We shall refer to the various rounds by the subscript j, and we shall call P the output of the permutation P. F has the same significance as in the ladder diagram. R and L refer to the contents of their respective registers and R_j, L_j to the contents after the jth round. The sub-key at the jth round is k_j.

During encipherment the following relationships are true

$$L_j = R_{j-1} \tag{1}$$

$$R_j = L_{j-1} +_2 P_j \tag{2}$$

$$P_j = F(R_{j-1},k_j) \tag{3}$$

where the operator $+_2$ has the significance of exclusive-OR. The second and third of these equations may be combined

$$R_j = L_{j-1} +_2 F(R_{j-1},k_j). \tag{4}$$

With equations (1) and (4) we now have a complete description of the jth state in terms of the $(j-1)$th state.

By simple rearrangement of equations (1) and (4) we can also write:

$$R_{j-1} = L_j \tag{5}$$

$$L_{j-1} = R_j +_2 F(R_{j-1},k_j) \tag{6}$$

Substituting from (5) in (6) we get

$$L_{j-1} = R_j +_2 F(L_j,k_j). \tag{7}$$

Thus from equations (5) and (7) we have a complete description of the $(j-1)$th state in terms of the jth state. These equations can be understood as describing decipherment from L_{16},R_{16} to L_0,R_0, using the sub-keys in the sequence k_{16},k_{15}, ... k_1. Note that L and R are interchanged — this is the reason for the swap of L and R at the end of the sixteen rounds of encipherment or decipherment.

Each distinct round constitutes an involution. In decipherment the inverse operation is achieved by taking the same set of involutions in reverse order.

3.3. THE EFFECT OF THE DES ALGORITHM ON DATA

It is the aim of the designers of the DES algorithm that plaintext data processed by the algorithm shall be transformed in such a complicated way that it is not possible to find any correlation between ciphertext and plaintext, nor is it possible to show any systematic relationship between ciphertext and encipherment key. The effect of a change of one bit in an input plaintext block should ideally be to change the value of each individual bit in the output ciphertext block with

a probability of one-half. There is little doubt that the designers have been successful in achieving this aim. To illustrate this assertion we shall now examine the way in which examples of plaintext blocks are enciphered by the algorithm. We propose to show that single bit changes of plaintext cause the ciphertext to be changed beyond recognition and that for the same plaintext single bit changes in the encipherment key produce totally different ciphertext.

In the figures that follow, plaintext, ciphertext and keys are all represented in hexadecimal notation. In Figure 3.14 a plaintext block is shown at the head of the table, with, later in the table, other selected blocks which each differ from the first by 1 bit only. All these blocks have then been enciphered under the same key (0123456789ABCDEF). A glance at the corresponding ciphertexts shows that the changes caused by the 1-bit differences in plaintext are large and random. To obtain some measure of the change the Hamming distance is listed of each of the subsequent ciphertext blocks from the first in the table; the mean Hamming distance from the first block measured over all the blocks is 31.06. This is very close to the expected value of 32.

In Figure 3.15 shows the result of enciphering the same plaintext block (ABCDEFABCDEFABCD) with a succession of different keys. At the top of the table we have encipherment with a chosen key, followed by encipherment with keys each of which differs from the first by 1 bit (parity correction has been included, so the appearance is given of a 2-bit difference). Against each subsequent ciphertext the Hamming distance from the first in the table is listed. Once again the mean Hamming distance of 32.88 is very close to expectation.

The DES transformation is thus seen to be efficient in obscuring similarities between plaintext. This can be illustrated even more graphically by examining the progression of the transformation within the algorithm. Figure 3.16 shows the result of encipherment of two plaintexts, which differ by 1 bit, under the same key (08192A3B4C5D6E7F). The contents of the L and R registers at the beginning of each of the sixteen rounds are listed (these quantities are called 'intermediate results'); the overall result of the encipherment transformation is also shown. The progression of the Hamming distance between the intermediate results at each stage can be traced (taking the combined L and R states within each encipherment process). For the first and second intermediate results the Hamming distance is low, but an appreciable distance soon develops and is maintained. By the completion of the fifth internal cycle there is very little correlation between the inputs and the intermediate results. There are many ways to examine this property and they all lead to the conclusion that five rounds would have been just sufficient to remove obvious regularities from the function. It seems that sixteen rounds are more than adequate.

3.4. KNOWN REGULARITIES IN THE DES ALGORITHM

Complementation

A notable feature of the DES algorithm is the property of complementation. If the complement of a plaintext block is taken and the complement of an encipherment key, then the result of a DES encipherment with these values is the complement of the original ciphertext.

Key 0 1 2 3 4 5 6 7 8 9 A B C D E F

	Plaintext	Ciphertext	Hamming distance
1	A B C D E F A B C D E F A B C D	C D E 8 7 2 D 4 A 4 7 1 3 4 6 F	
2	8 B C D E F A B C D E F A B C D	C D 3 D 0 A A 4 C 4 0 2 4 B 4 A	29
3	A 9 C D E F A B C D E F A B C D	8 0 1 F 8 A 2 9 6 8 B C 4 4 7 3	38
4	A B D D E F A B C D E F A B C D	5 D 9 8 C 4 7 D D D B A 6 F 3 0	36
5	A B C F E F A B C D E F A B C D	9 9 8 9 5 6 2 A 8 4 F 4 0 1 C 9	26
6	A B C D 6 F A B C D E F A B C D	6 7 C 2 6 9 F 2 5 4 2 7 9 1 F 9	30
7	A B C D E B A B C D E F A B C D	F 8 C 9 8 F 7 9 A D C 0 6 E A 4	33
8	A B C D F F D B C D E F A B C D	8 7 D 3 2 4 0 A B B F 4 4 0 7 4	34
9	A B C D E F A 9 C D E F A B C D	D B 9 9 8 B 6 7 0 4 6 C D C E 7	30
10	A B C D E F A B E D E F A B C D	2 F 6 E 5 4 7 0 E 4 E 3 5 1 A C	25
11	A B C D E F A B C C E F A B C D	B 5 3 E 4 2 D E 3 0 F 9 7 A D 0	29
12	A B C D E F A B C D 6 F A B C D	4 F 4 0 6 7 7 2 6 B 3 5 B 0 1 4	28
13	A B C D E F A B C D E 7 A B C D	A B 1 5 5 2 8 9 6 6 0 C 6 0 B 2	35
14	A B C D E F A B C D E F 2 B C D	5 B D A 9 3 F 7 D 4 2 7 B 8 D 2	30
15	A B C D E F A B C D E F A F C D	9 8 5 3 C 5 1 1 E D 5 6 8 8 7 E	34
16	A B C D E F A B C D E F A B D D	7 0 A A 2 4 0 7 9 5 9 F 0 4 B 1	34
17	A B C D E F A B C D E F A B C 5	8 9 2 B E C 4 7 C 9 7 1 2 B E 3	26

Mean Hamming distance 31.06

Fig. 3.14 DES encipherment of a series of data blocks each differing by 1 bit from a chosen block

	Plaintext	A B C D E F A B C D E F A B C D		
	Key	Ciphertext		Hamming distance
1	0 1 2 3 4 5 6 7 8 9 A B C D E F	C D E 8 7 2 D 4 A 4 7 1 3 4 6 F		
2	8 0 2 3 4 5 6 7 8 9 A B C D E F	1 B 7 3 F E 8 B C 0 B 8 8 6 0 6		35
3	0 2 2 3 4 5 6 7 8 9 A B C D E F	0 F 9 2 F 6 0 D 2 F D 4 D 8 B 7		32
4	0 1 6 2 4 5 6 7 8 9 A B C D E F	3 1 A F D 8 C 5 4 F B F 4 B C D		37
5	0 1 2 6 4 5 6 7 8 9 A B C D E F	C 7 9 F 5 9 6 3 D 4 6 5 A 7 E E		29
6	0 1 2 3 0 4 6 7 8 9 A B C D E F	3 6 5 2 9 C C 1 0 7 1 7 A 3 8 9		39
7	0 1 2 3 4 6 6 7 8 9 A B C D E F	7 F 3 5 F 7 E 6 C E C 5 7 E E 3		30
8	0 1 2 3 4 5 4 6 8 9 A B C D E F	C 9 F 3 F D 9 2 6 0 C 6 8 1 8 A		27
9	0 1 2 3 4 5 6 4 8 9 A B C D E F	E 6 9 2 8 3 2 2 E E 8 B 9 A 6 9		36
10	0 1 2 3 4 5 6 7 A 8 A B C D E F	0 9 9 7 A 5 A F 6 E 4 E 1 4 6 0		37
11	0 1 2 3 4 5 6 7 8 A A B C D E F	C 7 B 4 1 C 4 F 3 8 D 9 A F 7 A		31
12	0 1 2 3 4 5 6 7 8 9 E A C D E F	4 B 4 A A 2 0 A B 2 1 4 4 D D D		30
13	0 1 2 3 4 5 6 7 8 9 A 8 C D E F	1 2 9 9 7 E E 8 0 0 1 B D 2 7 C		32
14	0 1 2 3 4 5 6 7 8 9 A B 4 C E F	4 2 D 1 7 B 7 D F 5 3 4 3 B 7 9		28
15	0 1 2 3 4 5 6 7 8 9 A B C E E F	8 7 F 6 4 9 3 C 4 C 8 9 8 3 2 7		33
16	0 1 2 3 4 5 6 7 8 9 A B C D 6 E	F C 3 0 F 6 F 7 6 B D 4 2 5 9 2		31
17	0 1 2 3 4 5 6 7 8 9 A B C D E C	1 C B D E C 4 B 7 9 B C A 7 A 1		39
			Mean Hamming distance	32.88

Fig. 3.15 DES encipherment of a data block with a series of keys each differing by 1 bit from a chosen key

Encipherment with Key 08192A3B4C5D6E7F

Plaintext 1 0000000000000000 Plaintext 2 0000000000000001

Round	Register L	Register R	Register L	Register R	Hamming distance
1	00000000	00000000	00000080	00000000	1
2	00000000	AF0D68FD	00000000	AF0D687D	1
3	AF0D68FD	CE0A36EA	AF0D687D	CE3A32E2	5
4	CE0A36EA	0BDCC5FE	CE3A32E2	81BDED5F	15
5	0BDCC5FE	5D181CC3	81BDED5F	F8CA39B2	26
6	5D181CC3	1744B978	F8CA39B2	A994B918	26
7	1744B978	9B1CB0D8	A994B918	3E9D05D8	22
8	9B1CB0D8	7AE8C7E0	3E9D05D8	23F48DFF	26
9	7AE8C7E0	A2AC7B3F	23F48DFF	9D58DCFB	34
10	A2AC7B3F	580E51EF	9D58DCFB	3F076303	34
11	580E51EF	2865DBD4	3F076303	97AE4AE3	35
12	2865DBD4	BF68F77C	97AE4AE3	68ABB612	37
13	BF68F77C	8F57F629	68ABB612	09D9C398	32
14	8F57F629	0827B240	09D9C398	44B0435C	31
15	0827B240	F2DEBFAC	44B0435C	319FD4B8	29
16	F2DEBFAC	16CD0EB8	319FD4B8	42684FF9	24

Ciphertext 1 25DDAC3E96176467 Ciphertext 2 1BDD183F1626FB43 22

Fig. 3.16 DES decipherment of a block with two keys differing by 1 bit, showing the intermediate results before each round of the algorithm

Thus if

$$y = Ek(x),$$

then

$$\bar{y} = E\bar{k}(\bar{x}).$$

This result may seem surprising, especially when the non-linearity introduced by the S-box transformations is considered. The effect is entirely due to the presence of two exclusive-OR operations, one of which precedes the S-boxes in the logical flow of the algorithm and the other follows the permutation P. If the two input fields to an exclusive-OR operation are complemented, then the resulting output is exactly the same as that obtained for the uncomplemented inputs. Complementation of the plaintext and key inputs to the DES algorithm results respectively in complementation of the output of the expansion permutation E and the output from the sub-key selection procedure (following PC2); these two data fields are the inputs to the exclusive-OR before the S-boxes, so the S-boxes received unchanged data. Their unchanged output enters the exclusive-OR gate which transfers the (complemented) L register to the (complemented) R register.

The complementation property of the DES algorithm does not represent a serious weakness in the security of systems using the DES; there is no reduction in the time taken for an exhaustive key search, given a known plaintext–ciphertext pair. If a chosen plaintext attack can be mounted, then, by exploiting the complementation property, the time taken to exhaust the key domain by exhaustive

```
01 01 01 01   01 01 01 01
FE FE FE FE   FE FE FE FE
1F 1F 1F 1F   0E 0E 0E 0E
E0 E0 E0 E0   F1 F1 F1 F1
```

Fig. 3.17 The weak keys of the DES

search may be reduced by a factor of two, given the plaintext x and the values of $y_1 = Ek(x)$ and $\bar{y}_2 = Ek(\bar{x})$, so that $y_2 = E\bar{k}(x)$. If all values of x are searched to find if the $Ek(x)$ value equals either y_1 or y_2, then each test covers the two key values k and $\bar{k}$. If this is a matter for serious concern it might be said that steps should be taken to frustrate a chosen plaintext attack, probably by controlling access to the DES function. A system should not enable an opponent to discover both $Ek(x)$ and $Ek(\bar{x})$. From another point of view, it does not appear that any attempt has been made to exploit the complementation property in the design of applications of the DES.

The weak keys

A further regularity in the operation of the DES algorithm is due to the way in which bits are selected from the encipherment key to form the individual sub-keys, k_1 to k_{16}. Clearly, if the encipherment key bits are either all ones or all zeros, then the output from PC2 at each cycle will be the same for all rounds. Therefore, presentation of the sub-keys in the order k_1 to k_{16} will be identical with the order k_{16} to k_1. This means that encipherment and decipherment are exactly equivalent. Thus the effect of the algorithm on a plaintext block with a key of all zeros or all ones will be the same whether encipherment or decipherment is invoked. The result of this is that if a block is enciphered with either of these keys and then re-ciphered with the same key, the original plaintext will be restored; the function is an involution for these values of key. Futhermore, because of the way in which the C and D registers are segregated, having zeros in C and ones in D (or vice versa) will also produce a set of constant sub-keys. For this reason four encipherment keys which have this property can be specified. These are known as 'weak' keys[7] and are listed in Figure 3.17. They should be avoided when encipherment keys are being selected for critical situations, like master keys.

The semi-weak keys

In addition to the weak keys it can be shown that a further set of encipherment keys have a somewhat analogous property. These are the 'semi-weak' keys[8] which occur in pairs. For each pair of semi-weak keys the sets of sub-keys are identical but in the reverse sequence. Thus k_1 for the one encipherment key is equal to k_{16} for the other key of the pair, and so on. The result of encipherment with one key of a pair, followed by encipherment with the other key of the pair, is to restore the original plaintext. There are six pairs of semi-weak keys, which are listed in Figure 3.18. In the semi-weak keys, one or other of the C and D registers contains a pattern of alternate ones and zeros ... 0101 ... Their property is due to the symmetry of the shift patterns shown in Figures 3.10 and 3.11.

01	FE	01	FE	01	FE	01	FE
FE	01	FE	01	FE	01	FE	01
1F	E0	1F	E0	0E	F1	0E	F1
E0	1F	E0	1F	F1	0E	F1	0E
01	E0	01	E0	01	F1	01	F1
E0	01	E0	01	F1	01	F1	01
1F	FE	1F	FE	0E	FE	0E	FE
FE	1F	FE	1F	FE	0E	FE	0E
01	1F	01	1F	01	0E	01	0E
1F	01	1F	01	0E	01	0E	01
E0	FE	E0	FE	F1	FE	F1	FE
FE	E0	FE	E0	FE	F1	FE	F1

Fig. 3.18 The semi-weak keys of the DES

The semi-weak keys should also be avoided when selecting encipherment keys, though it is unlikely that they represent such a security risk as do the weak keys. Note that NBS uses the term 'dual keys' to embrace both weak and semi-weak keys.[2]

Further regularities in cycle keys are possible, in which some sub-keys are repeated in the sequence. In 1983 an account[7] was published in the journal *Science* of an alleged analysis claiming to show that patterns arising in the cycle keys led to serious deficiency in the strength of the DES algorithm. Here the term 'weak' was extended to include the keys we have called weak and semi-weak, together with other keys exhibiting regularity of cycle key pattern. It was asserted that cryptanalysis of the algorithm with any key was possible in 8 hours and that with the strongest of a general class of weak keys cryptanalysis should be possible in 4 hours (the 'worst-case' from the cryptanalyst's point of view). The 8-hour assertion is not dissimilar to that made by Diffie and Hellman and referred to later in section 3.5; note that this assertion demands the existence of a very large and powerful cryptanalytic machine. The size of the general class of weak keys was not defined in the published article, but it was said to include 25 different categories. Claims such as were made in the *Science* article would certainly be very damaging to any confidence that users might have in the security of the DES algorithm. We have seen no evidence to suggest that the claims made can be substantiated. In section 3.6 we discuss some of the academic studies that have been applied to the DES; none of the reported results makes claims remotely like those of the *Science* article. Indeed the paper by Moore and Simmons, discussed in section 3.6, gives an account of a detailed analysis of the significance of regularities in the cycle keys.

Hamiltonian cycles in the DES

There remains to be described a regular feature in the permutations of the DES algorithm.[8] We have seen how the strength of the whole algorithm depends on

the complexity of the function resulting from the iteration round the loop (R)ESP(R), where (R) represents the contents of the R register, E the expansion permutation, S the S-boxes and P the permutation of that name.

Consider now the transposition component of the functional loop, that represented by the E and P permutations. Since an arbitrary renumbering of the bits of both L and R is unimportant, it is not the individual permutations P and E that are significant but their product, the operation P followed by E (taken in this order because the path from S-box back to S-box takes them in this order).

The identification of the inputs and outputs of an individual S-box is also mostly insignificant. All the outputs are of equivalent status. The inputs, due to the nature of E, divide naturally into three classes. Let us denote the six input bits of an S-box as ABCDEF. The EF values of one S-box equal the AB of its neighbour to the right, for example in the case of S-boxes 1 and 2, E1 = A2 and F1 = B2. For our purpose, input bits A and B are not distinguishable because a permutation of the bits of the L and R registers could interchange any pair of these, interchanging also the corresponding EF pair.

The significant feature of the product of the P and E permutations is therefore not the bits they interconnect but the groups AB, CD and EF which they join. These can be tabulated as in Figure 3.19, showing for each of the S-box outputs which S-box input it affects, by way of the PE permutation. For example S-box output bit 1 of S1 affects F2 and B3 on boxes S2 and S3 respectively.

The way in which the outputs have been arranged in the DES obscured the pattern which is already evident, namely that of the four S-box output bits, two go round the loop to C or D inputs and two go to A, B, E or F inputs. The pattern can be shown best as a graph with directed arcs pointing from S-box output to S-box input. Figure 3.20 shows graphs separately for the CD and EF inputs. The nodes of the graphs are the eight S-boxes.

The graph in Figure 3.20 for CD can be expressed as two Hamiltonian cycles (87654321) and (84725163). The graph for EF has an identical form reflected about a vertical line. Both are very regular graphs which seem likely to have been a design intention. The apparent randomness of the permutation P is due only to the way that S-box inputs and outputs have been identified.

If the corresponding graph for the AB inputs is drawn on a pattern similar to those for CD and EF, then Figure 3.21(a) is produced which is much less regular than Figure 3.20. It can, however, be redrawn to show (Figure 3.21(b)), a less-confused structure.

The permutations in DES are a fixed feature, not subject to the key, and their purpose is to diffuse the effect of the substitutions so that a single bit change, if it expands to four potential bit changes after the first round, will (potentially)

S1				S2				S3				S4			
f2	f4	d6	d8	f3	e7	c1	c5	e6	e4	c8	c2	c7	e5	c3	f8
b3	b5			b4	a8			a7	a5				a6		b1
S5				**S6**				**S7**				**S8**			
e2	c4	f6	d1	e1	f7	d3	d5	e8	e3	c6	d2	f1	d7	d4	f5
a3		b7		a2	b8			a1	a3			b2			b6

Fig. 3.19 Interconnections between S-box outputs and inputs

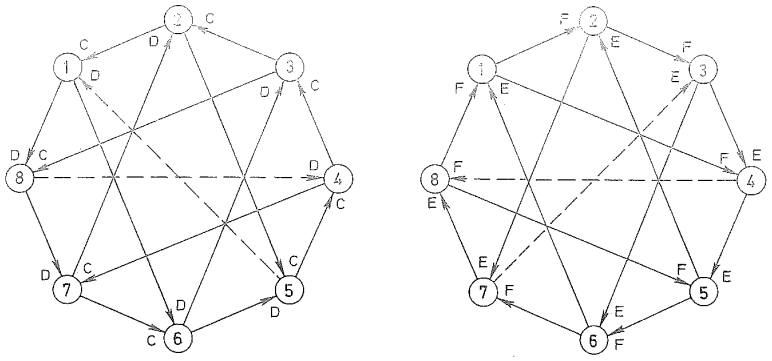


Fig. 3.20 Directed arcs for S-box CD inputs and EF inputs

affect the inputs to four S-boxes in the next round. We say (potentially) because it would be wrong to expect all the S-box outputs to change value — approximately half of them should do so. Also, some bit changes in the S-box outputs will affect two S-boxes on the next round, due to the expansion permutation. The aim of the permutation design (as reflected in our graphs) should be to spread the bit changes as rapidly as possible to all boxes. In this respect the CD and EF graphs are excellent, the AB graph less so. We can speculate that such considerations as these led to the remarkable regularity of two of the graphs and that the third graph is an unavoidable consequence of the first. Since they represent 'spreading' processes which operate in parallel, the good structure of two of the graphs is not impaired in any way by the third one. We understand that these permutating structures, though they were the consequence of a quest for good spreading, were not designed as regular structures except perhaps for the outer Hamiltonian cycles which are fairly obvious. Note also that our principle of ignoring the identity of S-box inputs and outputs, though it cannot be questioned for outputs, does cut across the stated design of the S-boxes as four separate bijections.

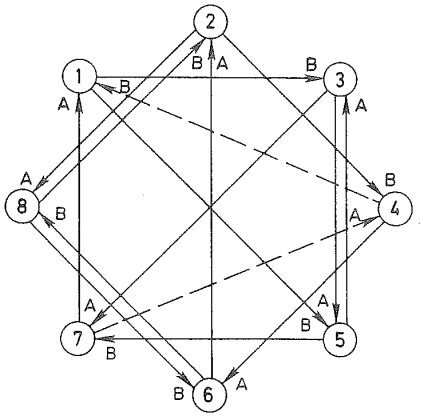


Fig. 3.21a Directed arcs for S-box AB inputs

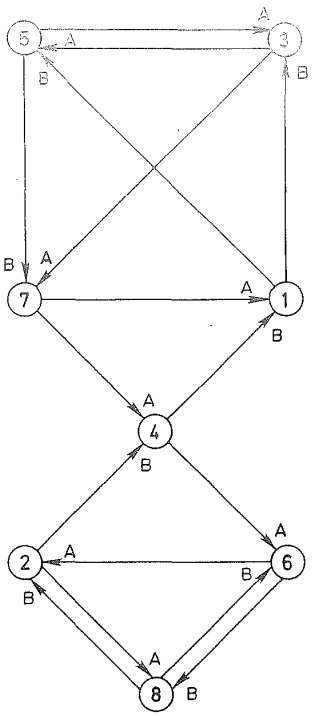


Fig. 3.21b Figure 3.21a re-drawn

3.5. ARGUMENT OVER THE SECURITY OF THE DES

When the DES algorithm was first published in the spring of 1975, a considerable amount of criticism was rapidly generated.[9] This was based on two premises — the length of the encipherment key was considered by some to be inadequate and it was alleged that 'trapdoors' or deliberate weaknesses might have been built into the S-box structures. These two criticisms are considered in turn.

Exhaustive search for a DES key

The length of the encipherment key is important when the security of the system against exhaustive search for the key is evaluated. For exhaustive search the cryptanalyst needs corresponding pairs of plaintext and ciphertext blocks; these may have been obtained by some breach of security in handling legitimate texts (a known-plaintext attack) or by deliberate introduction of text into the encipherment device by the opponent (a chosen-plaintext attack). If partial knowledge of the plaintext is all that is available to the cryptanalyst, perhaps in the form of knowing that the plaintext characters enciphered have odd or even parity, then his task is greater. In this case the search must list all the keys which yield possible plaintexts obeying the parity constraints; it is then necessary to repeat the

operation with other ciphertext blocks until the candidate keys have been narrowed down to one only.

Proceeding then on the assumption that the attacker has somehow obtained matching blocks of plaintext and ciphertext, the next step is to test all possible keys by enciphering the plaintext in turn with each and comparing the result with the known ciphertext; the process is known as a 'key search'. When a match is obtained between the produced and known ciphertext, then the current key value is that which is being sought. Clearly the time taken to exhaust the whole of the key domain will depend on the time taken to carry out the DES encipherment. If we assume that the time taken for the encipherment is 100 ms (the speed of the slowest microprocessor implementation known to us), that other processing times may be neglected, and that only one device is used, then we can calculate the time required to test all possible keys to be 7.2×10^{15} s, or about 228 million years. If we assume that the encipherment device is somewhat faster, with an operation time of 5 μs (a fraction slower than the fastest LSI implementation widely available), then the total time required to exhaust the key domain is just over 11 000 years. Clearly these times are totally impracticable for a cryptanalyst.

The exhaustive search times only become practicable when we postulate key searching which uses many DES devices in parallel, each of which is searching a different part of the key domain. Diffie and Hellman[10] have considered in some detail the design of a hypothetical machine constructed for this purpose. They assume that the operation time for each DES device is as low as 1 μs; this is justified on the dual grounds of advancing technology and economy of input/output. Since each special-purpose DES device need only be loaded once with the known matching pair of plaintext and ciphertext and with the encipherment key, then the overheads normally associated with input/output operations need not be taken into account. The only output from the device occurs when the required key has been found, when the fact of its recovery and its value will need to be output. Diffie and Hellman further assume that 1 million such DES devices can be assembled into one machine.

On these assumptions the time taken to exhaust the key domain can be recalculated; the time is reduced to about 72 000 s or just over 20 h. On average a key is found after testing half of the key domain, so the average time to find a key should be about 10 h. Evidently this time is low enough for users of the DES to be worried about the security of their data. We must, therefore, consider whether the assumptions made in arriving at this figure were reasonable.

Concern about the suggestion that an exhaustive key search machine was possible led NBS to hold a workshop 'on estimation of significant advances in computer technology' in August 1976.[11] This brought together experts who produced a range of estimates for the cost of the machine. The outcome from this meeting gave it a cost of $72 million and an expected availability date of 1990.

It was alleged that the mean time between failure of LSI devices is such that one could not run the machine without failure for more than a very few hours. Diffie and Hellman counter this by proposing a system of self-checking with an automatic switch-over to a reserve machine segment when failure is discovered. The cost of the machine has been estimated by Diffie and Hellman to be about $20 million, based on a unit cost for each DES device of about $10; IBM have put the cost at more like $200 million, based on a different unit cost. Diffie and

Hellman quote the total power consumption of the machine to be about 2 MW; another estimate that has been given is 12 MW.

On the assumption that the cost of the machine is amortized over a period of 5 years, Diffie and Hellman estimate that the cost per solution is about $5 000.

It is evident that the construction of an exhaustive-search machine is a very large undertaking, requiring very large resources. It has been conjectured[12] that the reason that the key length chosen was no more than 56 bits was in order that the key search machine should be viable. If the key length were 64 bits, an increase in key search times of a factor of 256 would be obtained; the parity bits do not seem to serve a particularly useful purpose. Against this it has been argued that excessive security is unnecessary for the intended field of application (unclassified, commercial, etc.) and that an increase in key length would increase the cost of the DES chips.

Multiple DES encipherment

It has at various times been suggested[13] that multiple-block encipherment under different keys would increase the security of enciphered data. It might be thought that an effective key length of 112 bits could be obtained by enciphering twice with different keys. However, a time-memory trade-off reduces the effective key length to 57 bits.[14] This is based on the assumption of a known plaintext attack in which the known plaintext is enciphered under all possible keys, then the known ciphertext is deciphered under all possible keys, giving two sets of data. These data are sorted to order and then compared one against the other, so that candidate key pairs can be identified. Note that the data unit for the sorting process consists of two elements — the result of the DES operation (encipherment or decipherment) and the key with which the result was obtained; this amounts to 120 bits per item. Since it is likely that a large number of candidate key pairs will be found (the theoretical number is 2^{48} false key pairs), it is necessary to check the candidate pairs against other pairs of plaintext and ciphertext blocks. Given a second known plaintext block and its ciphertext, the probability of finding a wrong key pair in addition to the correct pair is theoretically 2^{-16}. This attack has been called 'meet in the middle'.

Successive use of two encipherment keys can be made more secure by applying them in a more complex way. IBM use encipherment with key 1, followed by decipherment with key 2, followed by encipherment with key 1.[15] This is very much stronger and is used by IBM for protection of master keys in some of their cryptographic systems. A convenient feature of this system is that compatibility with single-key encipherment can be achieved by making the two keys have the same value. Even the degree of protection given by this triple operation has its critics,[14] Merkle suggests that a chosen plaintext attack will yield the two keys with about 2^{56} operations and 2^{56} words of memory. Note that this attack requires a chosen plaintext, whereas the attack on simple double encipherment with two keys requires only a known plaintext. Indeed the amount of chosen plaintext material required corresponds to 2^{56} DES operations; availability of this amount of material is so unlikely that the method is not practical.

Merkle suggests that a worthwhile increase in security can be obtained by triple use of the algorithm with three different keys (triple encipherment was put

forward as early as 1976 by Diffie and Hellman); this involves encipherment with the first key, decipherment with the second and encipherment with the third — if compatibility with the IBM two-key scheme is required, then this can be achieved by making the first and second keys of equal value.

Trapdoors in the DES?

The criticism of the DES algorithm has also concerned the choice of S-boxes. Hellman and others[16] have investigated the S-box structure and have shown that the security of a DES-like algorithm can be reduced by careful choice of S-boxes. By replacing the DES S-boxes by others of their own design, they have shown that it is possible to weaken the security of the encipherment while concealing the weak S-box structure to some extent. The assertion is then made that the actual DES S-boxes may themselves contain deliberate weaknesses put there in order to make the algorithm subject to a 'trap-door' attack by those in possession of the design information. On the other hand Hellman and his colleagues have not been able to demonstrate weaknesses in the DES S-boxes, though they point to certain unexplained systematic structure.

The permutation regularity which we discussed in the previous section of this chapter is one of construction. It does not, to our knowledge, result in any regularity of the DES function. The purpose of the permutations is to spread any change among the S-boxes in as few rounds as possible. For this purpose the EF and CD graphs are well chosen. For example a change in the input of S1 spreads to 6 and 8, and, from then on, the S-boxes affected in successive rounds are 3457, then 12345678, by which time all S-box inputs have been affected — on three rounds only. Since only half of the 64 bits circulating in the L and R registers are changed at each round, in practice six rounds are needed to spread a single bit change completely. The chosen graphs CD and EF are very efficient in this respect. Reference to Figure 3.16 where the intermediate data are shown for each round confirms this effect.

If the DES designers at IBM and NBS were able to set out their design criteria, then the kind of criticism which has been outlined in this section might well be allayed. However, at the request of NSA the design criteria have been withheld on the grounds that the designers came across principles of cryptographic design which were considered to be of importance to national security. The same ban on publication has been applied to the results of an IBM study of cryptanalysis of the DES, said to have occupied seventeen man-years.

Senate investigation of the DES

Because of these bans on publication, the controversy of criticism of the DES has been fuelled, rather than subdued. It has been alleged that NSA has exerted undue influence in the matter. Allegations of this kind led the US Senate Select Committee on Intelligence to investigate the role played by various government bodies during the design and development of the DES. The report of this committee is unavailable as classified material, but an unclassified summary[17] of its findings was issued in 1979. NSA was exonerated from tampering with the design of the DES in any way. It was said to have convinced IBM that a shorter key was adequate, to have indirectly assisted in the development of the S-box structures and

to have certified that the final DES algorithm was, to the best of their knowledge, free of any statistical or mathematical weakness. The Senate committee reported that the overwhelming majority of scientists consulted felt that the security of the DES was adequate for the time span for which its use was planned.

3.6. RECENT ACADEMIC STUDIES OF THE DES AND ITS PROPERTIES

During the public existence of the DES there has been a steady level of academic interest in its security and general properties; this has led to publication of a number of interesting results, some of which have only emerged relatively recently. A research group at the University of Louvain has published a series of papers,[18,19] giving an account of several investigations. In 1983 it was shown, using various equivalent representations of the DES algorithm, that is was possible to specify a totally iterative representation of the DES. It was also shown that the algorithm could be described with one fewer internal table and that it was related to the mixing transformation described by Shannon in his classical paper; a link was also established to nonlinear feedback shift register theory and to permutation networks. A successor paper in 1984 studied the degree of dependence of the output of the algorithm on its input, with particular attention being paid to small avalanche characteristics. The 1984 paper drew attention to some newly discovered properties of the S-boxes; the studies included tests with some S-box input bits fixed and observations made of the effect of varying the other inputs, whilst another test included complementation of certain input bits. Attention was also paid to the key scheduling part of the algorithm, reducing the number of internal cycles in order to simplify the study. Links between the S-box structure and the performance of the algorithm were identified. The paper is full of minor results, but the authors make no claim to have made cryptanalysis of the algorithm any easier; they do assert that a better understanding of the algorithm has been achieved.

Two papers[20,21] published in 1985 by the MIT group led by Ronald Rivest gave particularly interesting results. The general aim of the work was to determine whether the DES algorithm represented a group. The basis for this aim was the suggestion that if the set of permutations describing the DES transformation was closed under functional composition, then the DES would be vulnerable to a known-plaintext attack with 2^{28} steps on average, a possibly serious weakness. The results obtained by Rivest and his colleagues show, with an acceptable level of confidence, that the DES is not a group, therefore the basic supposition of DES vulnerability on this score is not substantiated. However, in the course of experiments carried out in this connection, a really surprising result was noted. The particular work related to cycling tests on the DES for keys selected in various ways, random keys and alternate encipherment with all zeros and all 1s as keys. A cycling test is designed to discover when, after a series of operations of the algorithm beginning from a recorded starting value, an output value is reached which has the same value as the start point. Encipherment with alternate keys of all zeros and all 1s produced a cycle length of rather less than 2^{33} operations; this was far less than that predicted according to analysis. It was estimated that the probability of such an occurrence was less than 1 in 10^9 if the permutation

was chosen at random from the symmetric group on message M. The result was very striking and caused much comment. The true explanation of the phenomenon was given by Coppersmith,[22] who showed that, given a weak key, there were 2^{32} fixed points for the DES; a fixed point represents a plaintext which is not altered by a cryptographic transformation — the ciphertext output of the algorithm is the same as the plaintext input. Each weak DES key has 2^{32} associated fixed points. If one pictures a cycling test as tracing a path through message space, it is immediately apparent that a fixed point causes this path to be reversed. It is the reversal of the path through message space that causes the start point to be reached much earlier than would be expected. Again, this result does not aid the task of the cryptanalyst. The discovery of the fixed points makes the case for avoiding weak keys in practical applications all the stronger.

Another interesting empirical result was reported by Shamir[23] in his paper given at Crypto '85; this showed some remarkable structural patterns in the DES S-boxes. If one divides the S-box outputs into those containing an even number of 1s and those containing an odd number of 1s (in other words a classification based on the parity of the outputs), the distribution of this classification in the S-box rows is very striking. In almost all S-box rows there is a bias of output parity either to the right or left half of the row; in most cases this bias is very marked, the phenomenon being most apparent in S-boxes 1, 5 and 8. The input element which selects the right or left half of an S-box row is the 'B' bit, using the notation expressed in section 3.4. The conclusion is that the 'B' input bit has a very strong influence on the parity of the output pattern. Whilst Shamir did not explain the significance of the phenomenon in his paper, he did consider that the unexpected result demonstrated the deficiencies of current certification techniques, arising from a lack of deep understanding of the nature of the DES algorithm. Having discussed the result with Brickell and Coppersmith, Shamir suggests that the observed properties of the S-boxes could be an unintentional consequence of some of the design criteria. Shamir also notes that the phenomenon was also reported independently by Franklin in a Berkeley MSc dissertation.

Further illustration was thrown on the effect of regularities in the DES cycle keys by Moore and Simmons[24] in 1987. Their study was prompted in part by the 1985 results of Rivest and his co-workers, followed by the interpretation put by Coppersmith on these results. Moore and Simmons confirm the explanation advanced by Coppersmith; they identify a class of cycles of the kind noted by Rivest and call these 'Coppersmith cycles'. A formal derivation is provided of the set of inverse DES key pairs that comprises the weak and semi-weak keys as defined earlier in this chapter. The related phenomenon of palindromic and anti-palindromic keys is investigated. Palindromic keys are those which give the same sub-key sequence whether they are read from sub-key 1 to sub-key 16 or from sub-key 16 to sub-key 1, in other words sub-key 1 is equal to sub-key 16, sub-key 2 to sub-key 15 and so on; only the weak keys have this property. Anti-palindromic cycle keys are those where sub-key 1 is the complement of key 16, sub-key 2 is the complement of sub-key 15 and so on; four of the twelve semi-weak keys have this property. Another interesting result is a proof of the assertion that each of the weak keys has exactly 2^{32} fixed points; similarly it is proved that the four anti-palindromic semi-weak keys have exactly 2^{32} 'anti-fixed' points, where the result of encryption is the exact complement of the plaintext.

The second part of the Moore and Simmons' paper is an extensive analysis of nine classes of cycles; it is beyond the scope of the present account to cover this further work, but it can be recommended to the interested reader. The main feature from this paper that we stress is that, apart from the specification of the fixed and anti-fixed points for eight specific keys, none of the results of the study seems to be of any use to a cryptanalyst faced with the task of deciphering a ciphertext which has been enciphered with an unknown key.

One of us (DWD) in an unpublished work has analysed an attack on the DES which exploits the correlation between two bits of S-box input in neighbouring S-boxes, due to the E permutation. In principle this provides inferences concerning the bits of the key if enough plaintext–ciphertext pairs are given. The information available from this method would amount to 16 bits of key and greatly weaken the cipher. The analysis of the effect shows that the quantity of plaintext–ciphertext pairs needed is comparable with 2^{56}, so the method does not improve on a systematic key search. There is slight evidence that the S-boxes might have been constructed to reduce this effect, but this is not conclusive. The analysis was extended to include three adjacent S-boxes, with the same result.

3.7. IMPLEMENTATIONS OF THE DATA ENCRYPTION STANDARD

Since the announcement and official launch of the DES in 1975 a number of manufacturers of microprocessors and LSI devices have designed, made and marketed devices embodying the DES. The performance spectrum has been extremely wide, ranging from inexpensive microprocessor implementations capable of carrying out the algorithm in 100 milliseconds to very fast LSI implementations which perform the algorithm in less than 5 microseconds. Some of the devices simply implement the algorithm in its basic block mode, whilst others also offer the more complex modes of operation which we shall encounter in our next chapter. The physical protection given to the devices also covers a very wide range, from simple chips with no protection to tamper-resistant boxes designed to withstand determined mechanical and/or electronic attack. There has therefore been available a very wide spectrum of equipment embodying the DES algorithm.

Not all the DES devices announced by manufacturers are still available. There is an observable tendency for system designers to choose fast LSI devices rather than the slower microprocessor implementations; the cost differential is no longer as great as it once was. Some of the slower implementations may not now be available as they are no longer considered commercially viable and have been withdrawn. Equally there is little doubt that some DES implementations have been extremely popular and these are found in many commercial systems for ensuring data security.

3.8. THE IBM CRYPTOGRAPHIC SCHEME

International Business Machines have developed a sophisticated data security system, using the DES, which caters for data in transit between terminals and

hosts (and between hosts) and in storage within hosts. This system, which is particularly noteworthy for its elaborate key management facilities, is discussed in chapter 6. It is an integral part of Systems Network Architecture (SNA) and may be used within the operating systems of IBM 360 and 370 computers. Within the host computers the DES capability is provided either by the Programmed Cryptographic Facility Program Product (i.e. in software) or by the 3848 Cryptographic Unit (a hardware device); the 3848 requires the Cryptographic Unit Support Program Product to control it. At the terminals the DES is implemented in special hardware.

The maximum data transfer rate of the 3848 is 1.5 Mbytes/s. The unit enciphers in CBC mode, using 64-bit data blocks. The encipherment operation is said to be faster than the input/output channel can handle data, so it appears that the equivalent block encipherment time is less than 5.3 μs. Since the unit is not intended for use on insecure sites, it is not heavily armoured. However, the master key storage is arranged to delete the keys if attempts are made to access this illegally.

A two-level key hierarchy is operated; the keys are classified as master and operational. Entry of master keys into the 3848 is carried out using the same design of key entry module as that used for the 3845 and 3846 tamper-resistant boxes. Entry of the master key into the 3848 requires operation of a physical lock. The master key is double length, 128 bits (112 effective), in order to enhance the protection given by master key encipherment to the operational keys. Operational keys are protected by multiple DES operations under the control of the two halves of the master key; if we call the latter ka and kb, each of normal length, then the encipherment of operational keys follows the sequence

Encipher under ka
Decipher under kb
Encipher under ka.

Compatibility of the 3848 device with single-key encipherment is obtained by giving identical values to ka and kb. Examination of the encipherment sequence just quoted shows that this has the effect of encipherment once with the identical key value; thus the security given to the operational keys is reduced.

The same scheme of treble encipherment with two keys has been adopted in the standards ANSI X9.17 and ISO 8732 described in chapter 6.

It is recommended that ka and kb be chosen by some random method such as coin tossing. The keys entered must have correct parity, as the 3848 is programmed to reject keys with incorrect parity; calculation of parity depends on the person generating the keys. As a further degree of protection the unit is programmed to reject any of the weak keys should these be input by the operator. As the IBM key management system uses variants of the master key, calculated by inverting selected bits, the unit also rejects variants of the weak keys. Tests for the semi-weak keys are not carried out.

3.9. CURRENT STATUS OF THE DES

We have discussed in this chapter the early development of DES in the field of US federal and national standardization. Before we leave the subject it is appropriate to consider the current status of the algorithm.

The algorithm has been reviewed and re-certified by the US federal authorities for use in the US private commercial sector every five years since its official launch in 1977.

A recent change of US policy towards the algorithm can be traced back to the issue in 1984 of US National Security Decision Directive 145 (NSDD-145). Under this directive a Systems Security Steering Group, with members from some of the major US departments of government, was enjoined by the US President to encourage, advise and assist the US private sector in achieving protection of some communications. The executive agent for this group is the Director, National Security Agency (NSA), acting as National Manager for Telecommunications Security and Automated Information Systems Security.

Arising directly from NSDD-145 a Commercial COMSEC Endorsement Program (CCEP), already implemented by NSA in the context of protection of government information, was made available to the private sector. Under this program communication security modules have been created that embody cryptographic algorithms designed by or for NSA; mechanisms have been set up for supplier approval. These NSA algorithms come in two types, I and II. Type I is used for the protection of classified government information. Type II was originally intended only for the protection of unclassified but sensitive national security information, but its application is now extended to cover the commercial private sector. Neither Type I nor Type II algorithms will be published, nor will either be available for export from the USA.

One may now ask where this leaves the DES and its users. It was originally intended that new equipment using the DES would not receive endorsement for government purposes after 1 January 1988; existing endorsed equipment was to be allowed to continue in use for an indefinite period. When this policy was announced it was perceived that it might cause problems for the US Treasury program for certification of electronic funds transfer authentication equipment. Indeed the US banks were known to be using DES very widely for protection of funds transfers of all kinds, some of which involved international traffic. Arising from concerns of this kind, discussions took place that resulted in a revision of policy regarding the DES. On 22 January 1988 the US Department of Commerce issued a press release in which the National Bureau of Standards reaffirmed use of the Data Encryption Standard for another five years, saying that it continued to be a sound method of protecting computerized data. However, for federal non-financial applications the DES must now be regarded as superseded by the algorithms developed under CCEP.

Fuller discussions of these developments can be found in Newman and Pickholtz[25] and in Howe and Rosenberg.[26] We shall consider the wider impact of this US government policy on the international use of DES in chapter 11 on security standards.

REFERENCES

1. Davis, R.M. 'The Data Encryption Standard in perspective', *Computer Security and the Data Encryption Standard*, National Bureau of Standards Special Publication 500-27, February 1978.
2. National Bureau of Standards *Guidelines for implementing and using the NBS Data Encryption Standard*, Federal Information Processing Standards Publication 74, April 1981.

3. Feistel, H. 'Cryptography and computer privacy', *Scientific American*, 228, 5, 15, May 1973.
4. Smith, J.L., Notz, W.A. and Osseck, P.R. 'An experimental application of cryptography to a remotely accessed data system', *Proceedings of the ACM Annual Conference*, 282, August 1972.
5. Shannon, C.E. 'Communication theory of secrecy systems', *Bell System Technical Journal*, 28, 656, October 1949.
6. National Bureau of Standards *Data Encryption Standard*, Federal Information Processing Standards Publication 46, January 1977.
7. Kolata, G. 'Flaws found in popular code', *Science*, 219, 369, 28 Januiary 1983.
8. Davies, D.W. *'Some regular properties of the 'Data Encryption Standard' algorithm'*, *Advances in Cryptology. Proc. Crypto '82*, 39, August 1982.
9. Branstad, D., Gait, J. and Katzke, S. *Report of the Workshop on Cryptography in Support of Computer Security*, National Bureau of Standards, Report NBSIR 77-1291, September 1977.
10. Diffie, W. and Hellman, M.E. 'Exhaustive cryptanalysis of the NBS Data Encryption Standard', *Computer*, **10**, 6, 74, June 1977.
11. Meissner, P. (Ed.) *Report of the Workshop on Estimation of Significant Advances in Computer Technology*, National Bureau of Standards, Report NBSIR 76-1189, December 1976.
12. Hellman, M.E. 'DES will be totally insecure within ten years', IEEE Spectrum, **16**, 7, 32, July 1979.
13. Diffie, W. and Hellman, M.E. 'Privacy and authentication: an introduction to cryptography', *Proc. IEEE*, **67**, 3, 397, March 1979.
14. Merkle, R.C. and Hellman, M.E. 'On the security of multiple encryption', *Comm. ACM*, **24**, 7, 465, July 1981.
15. Tuchman, W. 'Hellman presents no shortcut solutions to the DES', *IEEE Spectrum*, **16**, 7, 40, July 1979.
16. Hellman, M., Merkle, R., Schroeppel, R., Washington, L., Diffie, W., Pohlig, S. and Schweitzer, P. *Results of an initial attempt to cryptanalyze the NBS Data Encryption Standard*, Stanford University, Centre for Systems Research, Report SEL 76-042, November 1976.
17. 'Senate Select Committee on Intelligence' *New York Times*, 13 April 1978; *Computerworld*, 17 April 1978.
18. Davio, M., Desmedt, Y., Fossprez, M., Govaerts, R., Hulsbosch, J., Neutjens, P., Piret, P., Quisqater, J-J., Vandewalle, J. and Wouters, P. 'Analytical characteristics of the DES'. *Advances in Cryptology. Proc. Crypto '83*, 171, August 1983.
19. Desmedt, Y., Quisquater, J-J. and Davio, M. 'Dependence of output on input in DES: small avalanche characteristics', *Advances in Cryptology. Proc. Crypto '84*, 359, August 1984.
20. Kaliski, B.S., Rivest, R.L. and Sherman, A.T. 'Is the Data Encryption Standard a group?', *Advances in Cryptology. Proc. Eurocrypt '85*, 81, April 1985.
21. Kaliski, B.S., Rivest, R.L. and Sherman, A.T. 'Is the DES a pure cipher? (results of more cycling experiments on DES)', *Advances in Cryptology. Proc. Crypto '85*, 212, August 1985.
22. Coppersmith, D. 'The real reason for Rivest's phenomenon', *Advances in Cryptology. Proc. Crypto '85*, 535, August 1985.
23. Shamir, A. 'On the security of the DES', *Advances in Cryptology. Proc. Crypto '85*, 280, August 1985.
24. Moore, J.H. and Simmons, G.J. 'Cycle structure of the DES for keys having palindromic (or antipalindromic) sequences of round keys', *IEEE Trans. on Software Engg*, **SE-13**, 2, 262, February 1987.
25. Newman, D.B. and Pickholtz, R.L. 'Cryptography in the private sector', *IEEE Communications Magazine*, **24**, 8, 7, August 1986.
26. Howe, C.L. and Rosenberg, R. 'Government plans for data security spill over to civilian networks', *Data Communications*, **136**, March 1987.

Chapter 4 USING A BLOCK CIPHER IN PRACTICE

4.1. METHODS FOR USING A BLOCK CIPHER

Block encipherment operates on blocks of data of fixed size but a message to be enciphered can be of any size. Encipherment is sometimes applied to a stream of data, where the length of the stream may not be known when encipherment begins. So a block encipherment algorithm seems, at first, to have some limitations. Perhaps a stream encipherment algorithm might be closer to the real needs?

This reaction is probably wrong. A block encipherment method like the Data Encryption Standard (DES) can be used in various 'methods of operation' which cover all the practical requirements. Instead of being a limitation, the fact that the DES operates on fixed-length blocks makes it a convenient starting point for at least four different methods of operation which offer a very versatile set of 'tools' for encipherment.

Given a plaintext block x and a key k, the encipherment algorithm enables us to compute the ciphertext block $y = E_k(x)$. This, in itself, is a 'method of operation' sometimes called the 'native mode'. It can be applied to any source of plaintext in 64-bit blocks. This is actually a frequent requirement, for example when a DES key has to be enciphered. The DES key in its 64-bit form can conveniently be enciphered using the native mode of the DES algorithm.

A more general case is the encipherment of a message of arbitrary length perhaps made up of many 64-bit blocks. We can think of such a message as having been generated in a processor and stored ready for encipherment. We need a method for enciphering such a multi-block message which is convenient and secure. For this purpose there is a method of operation called 'cipher block chaining'.

The third case is the encipherment of a string of characters such as might come from a teleprinter which is being operated continuously. For such a stream we need a method of operation which enables each character to be enciphered as it arrives and does not have to wait until several characters have arrived before block encipherment can be carried out. This is a form of stream cipher and the related method of operation which will be described is called 'cipher feedback'.

There is a fourth requirement, which will emerge later in this chapter, for which another method of operation is used, called 'output feedback'.

The native mode of operation is called 'electronic codebook' because, for a given key, it associates with every value of the input block a unique value of the output block and vice versa. Because of the large number of such values, 2^{64}, the codebook in question is a very large one, never likely to be enumerated in full.

These four methods of operation have been found to cover most of the practical requirements for the use of encipherment in computer and network systems. They are usually called by the initial letters of their names and they can be summarized thus:

1. *Electronic Codebook (ECB)* the straightforward use of the algorithm to encipher one block.
2. *Cipher Block Chaining (CBC)* the repeated use of the algorithm to encipher a message consisting of many blocks.
3. *Cipher Feedback (CFB)* the use of the algorithm to encipher a stream of characters, dealing with each character as it comes.
4. *Output Feedback (OFB)* which is another method of stream encipherment with its own properties, to be described later.

These four methods of operation are the result of some careful study since the DES first became available. Other methods are possible but these four seem to meet most requirements. They are defined in a US Federal Information Processing Standard[1] and an international standard.[2]

The four methods can be used with any block cipher. In the descriptions which follow, the DES will be used because it is easier to describe the methods by a well-known example. It will be obvious how the methods can be used for any block cipher, changing only the details according to the block size.

The limitations of the electronic codebook mode

It might seem that a message comprising many blocks could be enciphered simply by dividing it into 64-bit blocks and enciphering each one separately. This is a simple application of ECB encipherment — the native mode. Decipherment is equally obvious. But, in general, ECB encipherment should not be used in this way because of some severe security weaknesses which arise from the structure of typical messages.

In all practical applications, fragments of messages tend to repeat. One message may have bit sequences in common with another. Messages generated by computers have their own kind of structure. The need for formal definition of protocols and the desire for generality in their definition has made large parts of the formats used by computer messages very repetitive. They may contain strings of zeros or spaces. The protocol designer provides parameters which in practice are rarely used and, therefore, take constant values. Computers often generate messages in a fixed format, having the important data always in the same place.

If the ECB type of encipherment is used with this kind of traffic, an alert enemy who intercepts collections of these messages will be able to detect the repetitions. Figure 4.1 illustrates how ECB encipherment fails to conceal this structure.

Repeated phrases will not show through the ECB encipherment unless they happen at the same phase relative to the block size. Cryptanalysts will search for and exploit the occasional visible regularities. The commonest repeats, such as strings of zeros, show up easily. The biggest danger arises when the significant parts of the messages change very little and appear in fixed locations. Analysing these parts becomes a 'codebook' exercise in which the number of code values is small.

Suppose now that a set of messages has been analysed and some partial

Block	Plaintext	Ciphertext
1	T H E Y C A N	60 99 46 42 52 82 22 49
2	H A V E S E	FF BF BC 77 8B BB F2 06
3	V E R A L A C	0D 4D 86 DE B6 CD 92 5D
4	T I V E P E R	99 63 A8 OF 32 D3 E7 E9
5	M A N E N T V	10 49 1F 3B DE 67 21 B7
6	I R T U A L C	BD 2D 6D 61 42 08 C7 B8
7	I R C U I T S	19 F1 01 A4 89 6A AE 4C
8	A N D / O R V	84 DB CC EC 35 18 58 9C
9	I R T U A L C	BD 2D 6D 61 42 08 C7 B8
10	A L L S A T	D4 3C D4 5A 9E 0B A5 ED
11	T H E S A M E	84 52 01 AC 2D FE 98 3A
12	T I M E .	89 F1 89 E9 DB CC CB BB
	Key	01 23 45 67 89 AB CD EF

Fig. 4.1 A weakness in ECB encipherment

understanding has been obtained of these parts of the messages. An enemy can re-construct messages from the parts of the message he has intercepted. The ECB mode of operation has not linked these blocks together, so they can be reassembled in any order. For example, a bank payment message might have the payment amount or the payee account number in fixed positions. Even if the enemy does not know the precise amount, he might be able to deduce the payee. He may also be able to find which are the large payments. He could exploit his knowledge of the enciphered blocks by forging large payments for chosen accounts, simply by reconstructing messages from known pieces.

The vulnerability of systems to 'codebook' analysis is greatest at the beginning of messages, where formal headers appear which contain the destination, serial number, time etc. The problem of the 'stereotyped beginning' of messages has to be considered carefully in all cipher systems.

The weakness of the ECB method lies in the fact that it does not connect the blocks together. By enciphering each block separately it leaves them as separate pieces which the cryptanalyst can analyse and assemble for his own benefit. What we need is a method of enciphering successive blocks which makes the cipher meaningless except in the given sequence. This is called 'chaining'.

4.2. CIPHER BLOCK CHAINING

Cipher block chaining uses the output of one encipherment step to modify the input of the next, so that each cipher block is dependent, not just on the plaintext block from which it immediately came, but on all the previous plaintext blocks.

Figure 4.2 illustrates how the method operates. All the operations in the figure work with 64 bits in parallel. In a real system, some of these transfers might be serial but the principle is best explained as for parallel operation throughout.

The operation of addition in the figure is the 'modulo 2 addition' function and it operates in parallel on the 64 bits of the blocks. The encipherment method, shown on the left of the figure, has a feedback path from the ciphertext output back to plaintext input. Except for the first block, each successive block, before encipherment, is added to the ciphertext of the previous one and this makes the nth ciphertext block C_n a function of all the plaintext blocks $P_1, P_2, \ldots, P_n$.

The figure shows on the right-hand side how this process is reversed for decipherment. After decipherment of the ciphertext, a correction is made which corresponds exactly to the operation carried out at the beginning, namely the addition, modulo 2 of the *previous* ciphertext block.

The words 'addition, modulo 2' do not adequately express the function shown by the circles in the figure. In fact, each bit of the input block P_n is added, modulo 2, to the corresponding bit of the previous ciphertext C_{n-1}. In actual implementations, the operations may be serial, or be carried out on one octet in parallel, according to the speed required or the DES implementation employed. The figure shows the operations as though they take place with 64 bits in parallel.

In order to make quite clear the sequence of operations, the ciphertext block is shown at both ends as entering a register where it remains between successive operations. This clarifies the point that, at both ends, the quantity entering the modulo 2 addition is *the ciphertext of the block which preceded the one now being processed.*

The operation of the cipher block chaining can be expressed as follows

encipherment:

$$C_n = Ek(P_n +_2 C_{n-1})$$

decipherment:

$$Q_n = Dk(C_n) +_2 C_{n-1}$$

In order to prove that the result of decipherment, Q_n, is indeed the plaintext, apply the operation Dk to the first equation, giving

$$Dk(C_n) = P_n +_2 C_{n-1}.$$

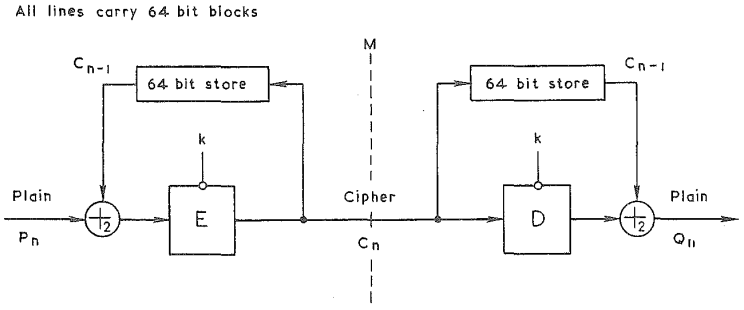


Fig. 4.2 Cipher block chaining

Substituting this value in the second equation

$$Q_n = P_n +_2 C_{n-1} +_2 C_{n-1} = P_n.$$

There is a more graphic way to demonstrate this inverse property which derives from the symmetry of Figure 4.2. It has 'mirror' symmetry about the broken line marked M, except for the data flows. The flow of data through the encipherment/decipherment operations is consistent with a symmetric pattern of data on the lines, noting that modulo 2 addition is naturally symmetric because the flow of data on the horizontal line can be reversed without changing the data values. It follows that the right- and left-hand sides are symmetric about the line M in the data values carried by the lines. From this it follows that the input and output data are the same. This kind of argument is very useful for establishing quickly the plausibility of systems employing a block encipherment algorithm, though it should be checked by a more rigorous analysis and the timing or sequencing of the operations should be checked. The 'mirror' argument is an appealing one because it can be seen directly in a figure and we shall use it again.

The first and last blocks

To complete the definition of CBC operation, both the start and the finish of the procedure must be detailed. At the beginning, the 'previous ciphertext block' is not available so the initial contents of the registers in Figure 4.2 must be specified otherwise. This is the quantity shown in the following equations as I. It is known variously as the 'initializing variable' or 'initialization vector'. We shall call it the *initializing variable (IV)*. The choice of this value is important to the security of the operation and it must be the same for the sender and receiver. The equations for expressing encipherment and decipherment must be replaced, for $n=1$, by

$$C_1 = Ek(P_1 +_2 I)$$
$$Q_1 = Dk(C_1) +_2 I$$

and the equations given earlier hold for $n=2,3, \ldots$.

When cipher block chaining is used in a communication system it is normal to keep the IV secret. This can be done by transmitting the IV encipherment in ECB mode using the same key that will be employed for data encipherment. More important than the *secrecy* of the IV is its *integrity*. We want to prevent an enemy from being able to make purposeful changes to the plaintext which emerges from the encipherment system. With CBC encipherment this cannot be done by manipulating the ciphertext because changes to the ciphertext will produce an apparently random effect in the plaintext output. But changes to the initializing variable would have the effect of making corresponding bit changes to the first block of plaintext output. In this way the whole of the first block is vulnerable to purposeful changes by an enemy. The problem is avoided by sending the IV in enciphered form.

Given a message of arbitrary length, it may not divide neatly into 64-bit blocks but leave something at the end. We can deal with the short end block in several ways but the simplest is to pad out the message by adding some extra bits until the block reaches the correct size. These padding bits could be zeros, because the addition of the previous ciphertext block before encipherment prevents them

being used for codebook analysis. On the other hand, the prudent designer might consider padding out with random digits just to make the analyst's job harder.

The use of padding gives a new problem to the system designer because the amount of padding has to be indicated somewhere so that the receiver can remove it. One convention is that the last octet of padding indicates by a 'padding indicator' (PI) the number of padding octets, as shown in Figure 4.3. This method requires that the existence of padding be revealed earlier in the message format. In a format that has been designed with encipherment in mind, special provision for a padding indicator might be made.

For many applications the system designer must have a method of handling the short end block which does not expand the message size. This would be needed, for example, when a stored plaintext is being enciphered and put back into the same store. For this purpose the scheme shown in Figure 4.4 is useful.

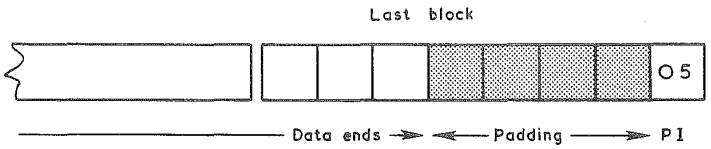


Fig. 4.3 Padding in the last block

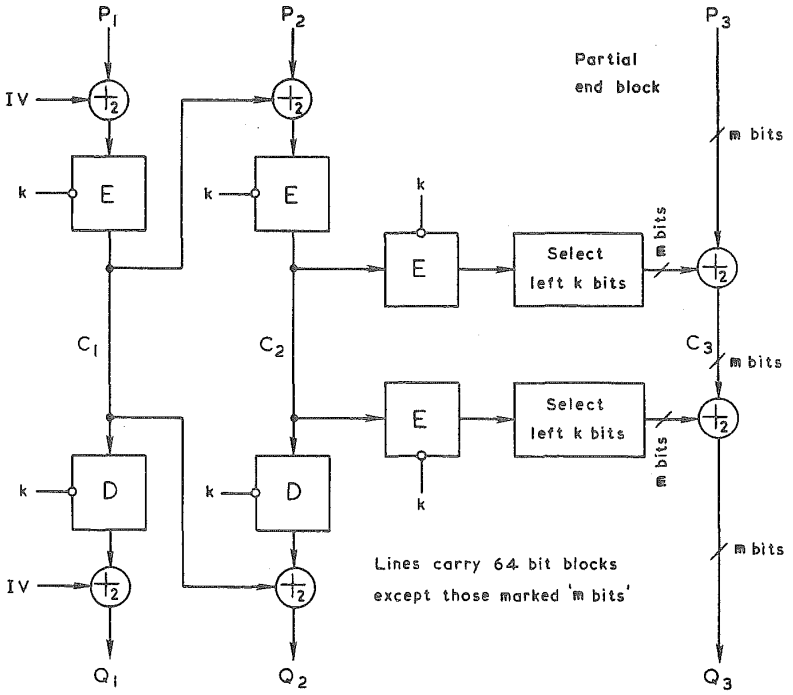


Fig. 4.4 A method for enciphering the end block

This figure shows the same basic procedure as Figure 4.2, but the treatment of each block of the message is shown separately. It shows the case of a message of two complete blocks followed by one partial block which has m bits. The final ciphertext block from the cipher block chaining process is enciphered once more and used by modulo 2 addition to treat the last, short block. Its value is sufficiently random for this purpose. The first m bits of the generated pattern are used to add, modulo 2, to the short block for transmission. A similar process at the receiving end generates the same pattern of m bits which deciphers the short block by modulo 2 addition.

There is a weakness in any such additive encipherment. Although the enemy may not know what is being enciphered, he can change it systematically. He can choose individual bits in the last plaintext block and change their values simply by changing corresponding bits in the ciphertext. This is not dangerous unless the part of the message we are talking about contains essential information and fortunately this is not usually the case. The end of a message usually carries sum checks which cannot usefully be changed. When used with care, this method of dealing with the short end block can be effective.

Transmission errors in CBC encipherment

The CBC method of operation can be characterized as *feedback* of ciphertext at the sending end and *feedforward* of ciphertext at the receiving end. The feedback process has the property of extending indefinitely the effect of any small change in the input so that a single bit error in the plaintext message will affect the ciphertext from the block in which it occurs and every succeeding ciphertext block until the chain ends. This is not as important as it may seem because the decipherment process restores the correct plaintext except for the error.

Errors in the ciphertext are more important and may easily result from a noisy communication path or a malfunction in a storage medium. The effect of a single bit ciphertext error on the received plaintext is shown in Figure 4.5.

The first effect is to change the input to the DES algorithm at the receiver. The output of the algorithm is then, for all practical purposes, random, because of the good cipher properties of the algorithm. At the time of the next block, the ciphertext error emerges from the register in the feedforward path and a single

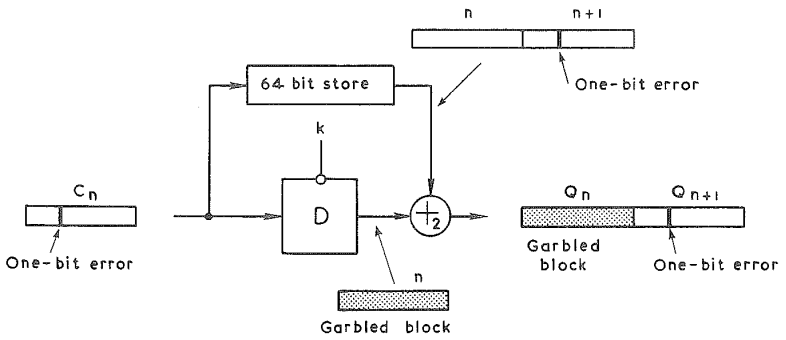


Fig. 4.5 One-bit error in cipher block chaining

bit error in the ciphertext causes a single bit error in the plaintext, in a corresponding position of this second block.

Blocks after the second are not affected by the ciphertext error, so in this respect the cipher block chaining procedure is self-recovering. That is to say: two blocks in the plaintext output will be affected, but the system recovers and continues to work correctly thereafter.

In contrast to the rapid recovery from bit errors, the system does not recover well from synchronization errors. If it is possible for a bit to be lost or gained in transmission so that blocks are now shifted one bit out of position, then the receiving system will generate garbage indefinitely. The method of using cipher block chaining must ensure that the framing of blocks cannot be lost indefinitely. This will generally be achieved when it is used for data in storage or in conjunction with data-link protocols that use framing.

The property of converting brief errors in transmission into lengthier errors of plaintext output is called 'error extension' or 'garble extension'. It is a nuisance in communication systems and may play havoc with a nicely tuned error-control procedure that depends on the statistics of error occurrences. For this reason, error control should be applied directly to the ciphertext stream, not around the larger plaintext loop which includes encipherment and decipherment. Figure 4.6 shows these two alternative locations of error control.

The error properties of CBC carry a security risk if they are not handled correctly. Errors randomize one block but they cause controlled changes to the *next* block. A system which ignored the errors in one block (or allowed them to be recovered later) and accepted the succeeding block would be vulnerable to calculated changes being made by an enemy to the ciphertext.

For this reason, although CBC recovers fairly quickly from line errors, the way in which it is used should abort any chain in which errors have occurred. Error recovery procedure should be designed to repeat the whole chain and to accept it only when it is all delivered correctly. For this reason, it is normal to provide a sum check on the plaintext to detect these errors at the receiver. Any enemy trying to modify bits in the chain would probably cause a random sum check and this would be detected. In chapter 5 this is treated more fully.

Choice of the initializing variable

We described cipher block chaining as a method by which the blocks could be linked together to prevent codebook analysis. This is achieved for all the blocks after the first, but the encipherment of the first block cannot depend on earlier blocks. If the initializing variable is kept constant for a series of chains, the repetition of patterns in the beginnings of the chains will show through. Using the same IV and key, two messages which have the same plaintext string at the beginning will have the same ciphertext until the point at which the two messages diverge. This degree of transparency would be very unwise, so we must contrive that either the beginning of the message or else the value of the IV differs on each occasion.

For communication systems it is customary to use as chain identifier a serial number at the start of the chain which does not repeat during the currency of

a key. On the other hand, there are applications of cipher block chaining where this message expansion is not acceptable, such as data enciphered *in situ* in storage. For these, it is the IV that must be made variable. For example, the IV can be derived from the index used for looking up the data in storage.

4.3. CIPHER FEEDBACK

Data are handled in many forms, as complete messages, or sequences of frames, blocks, 8-bit characters or binary digits. When messages must be treated character by character or bit by bit, another kind of chained encipherment is used which is known as *cipher feedback.*

When we treat data as a stream of bits or characters we are at a low level in the hierarchy of protocols which makes up the computer network architecture. This is a level at which transparency is important. We must introduce encipherment in a way which disturbs the existing system as little as possible. For example, consider a character terminal which sends out a stream of characters in the ISO 7-bit code with an added parity bit. Each of these octets from the terminal must be transmitted through a communication network at once and encipherment must not cause a delay. So the encipherment method, although it chains these octets together, must operate on each octet as it arrives and deliver it, enciphered, as soon as possible. The same requirement for immediacy applies to the decipherment of the octet stream coming into the terminal.

The cipher feedback method is illustrated in Figure 4.6. Superficially it appears similar to cipher block chaining but the important difference is that the DES block encipherment operation takes place in the feedback path at the transmitting terminal and in the feedforward path at the receiving terminal.

The encipherment of the character stream consists of adding, modulo-2, the character derived from the output of the DES algorithm to the plaintext character to form the ciphertext character and then at the receiving end adding the same data into the ciphertext character to restore the plaintext. The design of the system makes this added data stream pseudo-random and yet provides the same stream at the sending and receiving ends.

Whereas cipher block chaining operates on whole blocks, cipher feedback

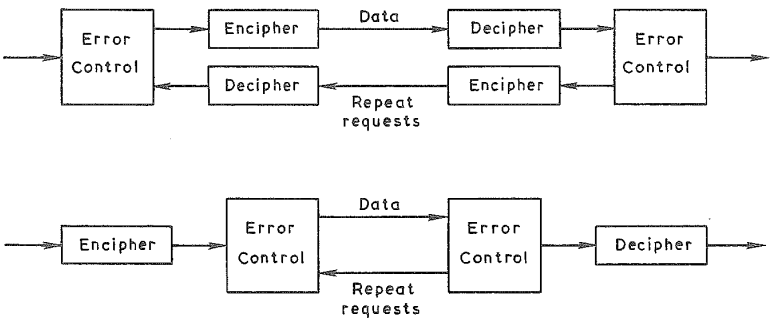


Fig. 4.6 Alternative locations for error control

operates on one character at a time, and the character length m can be chosen as a parameter of the design. It is known as 'm-bit cipher feedback'.

The smallest character length is 1 bit, resulting in '1-bit cipher feedback'. In principle, any value of m could be used up to 64. The most usual choice of m is the character size used in the communication system. (In older systems this can be 5 or 6 bits and in modern systems 7 or 8.) Probably the most frequent choice of character size today is the octet and Figure 4.7 is illustrated with this character width.

The eight bits used to encipher the character stream by modulo 2 addition are derived from bits 1 to 8 of the output of the DES algorithm. It is important to realize that the DES algorithm is performing an *encipherment* at both ends of the line. The input of this algorithm comes from a 64-bit shift register which contains the most recent 64 bits transmitted as ciphertext. Every character which goes out on the line at the sending end is shifted into the high-numbered bits of the shift register, displacing a similar number of bits from the other end. The same thing is done to the received ciphertext at the other end of the line. The argument of symmetry which was useful in understanding cipher block chaining can be applied to Figure 4.7, but the layout of the figure is intended to make clear the numbering of bits into and out of the DES algorithm and this rather obscures its symmetry.

If the ciphertext is transmitted without error, the contents of the shift register are the same at both ends. Therefore, the output of the DES algorithm is the same at both ends and the same eight bits are added to the data stream. This ensures that encipherment is correctly reversed by decipherment at the receiving end.

The sequence of operations as each character passes through the system is as follows. The incoming plaintext character has added to it, modulo 2, the bits coming from the DES algorithm. By this addition the character is enciphered and made ready for transmission. At the same time as it is transmitted it is loaded into the shift register, displacing the same number of bits from the left-hand or low-numbered end. Now the shift register contains a new pattern of bits and the DES algorithm is invoked to produce a new output. This output is ready for encipher-

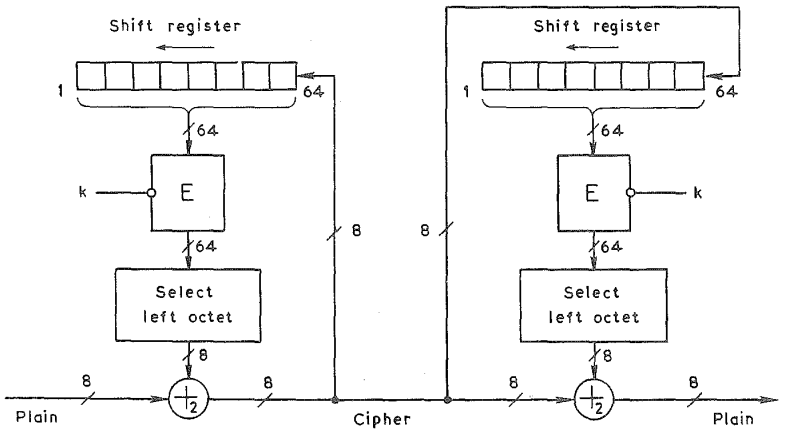


Fig. 4.7 8-bit cipher feedback

ing the next character. A similar procedure takes place at the destination where the incoming characters are deciphered (by adding bits from the DES output) and also fed into the shift register to produce the same stored pattern as at the sending end.

Each passage of one character through encipherment or decipherment invokes the data encipherment algorithm once. This contrasts with cipher block chaining where each operation of the algorithm enciphers 64 bits of users' data. For this reason, cipher feedback throughput could be limited by the speed of the DES operation. Fortunately, the lower levels of network hierarchy, at which cipher feedback is used, are often quite slow in operation so this speed limitation is not important.

Like cipher block chaining, cipher feedback chains the characters together, making the ciphertext a function of all the preceding plaintext.

Error extension in cipher feedback

CFB operation is like cipher block chaining in having a feedback path at the sending end and a feedforward path at the receiving end. Correspondingly, changes in the plaintext affect all the succeeding ciphertext, and errors on the communication line are subject to 'error extension'. We can follow the effect of a single bit transmission error in Figure 4.8. The first effect is to alter the corresponding bit in the plaintext output of that character. The error also enters the shift register, where it remains until the shift process pushes it off from the left of the register. While the error is in the shift register, the output is garbled. In the case of 8-bit cipher feedback which is illustrated, nine plaintext characters are affected by a single error. The error properties are similar to those of CBC except that the 'controlled' change takes place at the beginning of the error pattern instead of the

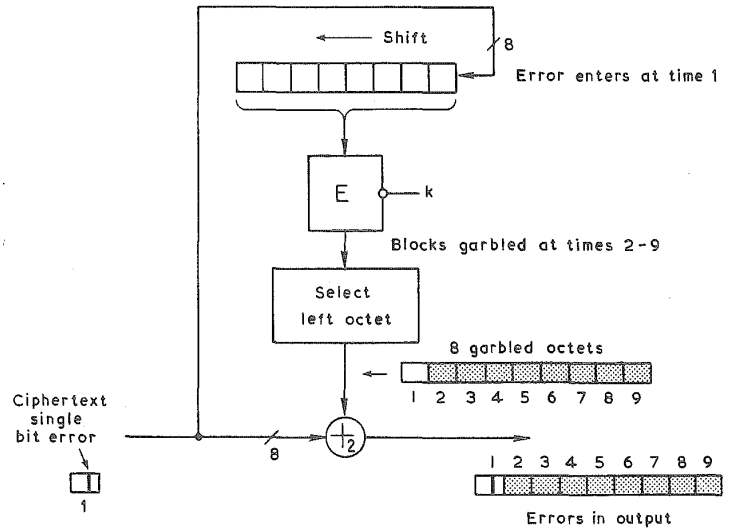


Fig. 4.8 One-bit error in 8-bit cipher feedback

end. An enemy could change selected bits of a character in the plaintext while completely garbling the following eight characters.

When the purpose of encipherment is simply to avoid disclosure this error pattern is a nuisance but no more. For an active attack it could be a weakness because the first character of the error pattern can be modified purposefully. Usually this trick would be detected because of the following garble, which a plaintext sum check would reveal as an error. A dangerous case occurs with the last character of a message, where the garbled characters never come. This matters little for 1-bit CFB and is unlikely to matter for 8-bit CFB since the 8 bits in question are probably part of the sum check, but it could be serious with 64-bit CFB and, for example, a 16-bit sum check. The enemy could alter the last 48 bits of the message and correct the sum check to match. Of course, 64-bit CFB is unusual and CBC is likely to be used instead.

Initializing with CFB

To start up the process of cipher feedback, the shift registers must be loaded with an IV. It is customary to use a different IV for every chain during the lifetime of a key, in order to mask any repetitions at the beginnings of the chains. These IV values can be transmitted in clear form because they enter the process through the encipherment algorithm.

In cipher block chaining the transmission of the IV is a procedure quite distinct from normal message transmission because the IV is enciphered. The plaintext IV which is sent at the start of CFB operation can be regarded as a preamble to the message. It enters the shift register at both ends of the line and, when the registers are loaded with the same data, secure transmission begins. This method of start-up is characterized by random data being produced at the sending end (to conceal any regularities in the starts of messages) and different random data being produced at the plaintext output of the receiving end. In effect, the self-correcting properties of this cipher method are being used for starting up.

More formal procedures for initializing are described in chapter 11, in which a randomly chosen IV is sent before transmission of data begins. Each chain incurs this transmission overhead. There are polling systems which use very short messages, each message sent to a different station on a multi-station line. If CFB encipherment is used with a different key for each station, then each of these short mssages must be preceded by an IV. To avoid the full-length, 64-bit IV, it is customary to send a shorter-length value and pad it on the left (most significant) end with zeros to make up the 64-bit pattern in the shift register. Of course the transmitted value should not be too short or the danger of code book analysis of the starts of messages returns. For example, if an 8-bit IV is transmitted the enemy has to construct 256 versions of the codebook. Suppose that the polling message consisted of an address which was fixed for the station followed by a command with only a few possible values, then it would not take long to analyse these command values for a given station.

Encipherment of an arbitrary character set

Cipher feedback is used for enciphering a stream of characters, where each character is represented by m bits. In such a character set a character is represented

by a number in the range 0, 1, ... $n-1$ where $n=2^m$. Both the plaintext and the ciphertext characters are in the same range.

Some character codes do not occupy a range of 2^m values. For example, we might want to use the digits 0, 1, ... 9. Some character sets must avoid those binary values that represent control functions. For example the ISO 7-bit coded character set[3] (which is similar to the ASCII code) includes character values which mean: start of heading, start of text, end of text, acknowledgement, data link escape etc. In many communication systems, if we tried to send such a character in a ciphertext stream it would have an unfortunate effect on the network's procedure, so we have to restrict the transmitted characters to a 'safe' set. In the ISO case this usually means the binary values representing the numbers 32 to 127, ninety-six values in all.

We could consider an arbitrary character set which does not even use a consecutive set of character values. Then the first step in encipherment would be to map these characters into the consecutive values 0, 1, 2, ... $n-1$ where there are n different character values in the allowable set. If $n=2^m$, normal CFB operation can be used. Otherwise m bits are needed to encode the character where m is the smallest value for which $n<2^m$.

Figure 4.9 shows how the cipher feedback method of operation can be modified for the case of an arbitrary character set.[4] The process of encipherment in CFB is the addition of a stream of m-bit characters coming from the least significant m-bit positions of the DES output into the plaintext character stream. For dealing with the arbitrary character set modulo 2 addition of bits is replaced by modulo n addition of character values. Both these methods of encipherment are well established in classical ciphers, where binary addition, modulo 2, characterizes the Vernam method and modulo n addition characterizes the Vigenère method.

Cipher feedback has the useful property that the method of deriving the character stream for encipherment can be chosen almost at will, provided the same method is used by the receiver for decipherment. We must now show how a character in the range 0, 1, 2, ... $n-1$ can be produced from the output of the DES encipherment.

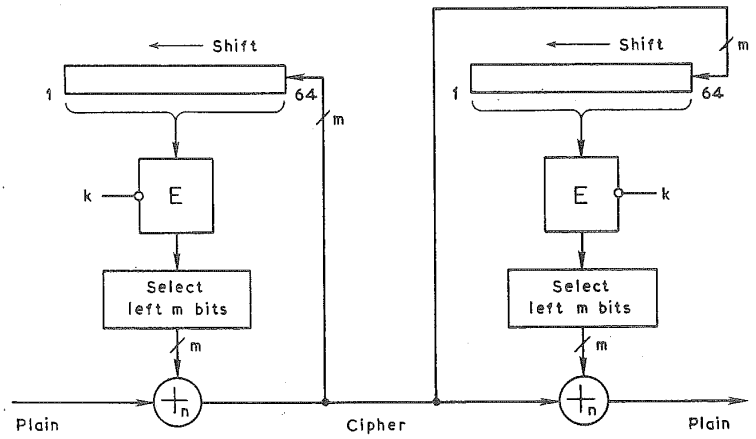


Fig. 4.9 Modulo n cipher feedback, $n<2^m$

The simplest proposal is to use the least significant m bits of the DES output where

$$2^{m-1} < n < 2^m.$$

The character value produced by these m bits could be added, modulo n to the plaintext to generate the ciphertext.

Figure 4.10 illustrates the procedure for deriving the ten character values 0, 1, ... 9 from a 4-bit output taken from the DES. This method is not recommended because the values 0, 1, 2, 3, 4 and 5 will occur twice as often as the values 6, 7, 8 and 9. If the digits are not equiprobable in the plaintext there will be a distorted distribution in the ciphertext, which can be exploited by a cryptanalyst. For general use we need the additive character stream to give all the values 0, 1, ... $n-1$ with equal probability.

A useful method is illustrated in Figure 4.11 for the example of generating decimal digits. The 64-bit output of the DES is divided into sixteen blocks of 4 bits each. Starting from the least significant end, the first four bits are examined and if they lie in the permitted range 0 to 9 this value is used for the additive encipherment. If not, the next block is tested in the same way. Since there are sixteen possibilities, it is highly probable that one of them will generate a satisfactory character and the first one to be reached is the one used for encipherment. If, unfortunately, all the values happen to lie in the range 10 to 15 then we choose some arbitrary prearranged value to add to the character for encipherment, say 5 for example. The probability that this arbitrary value will have to be used is $(6/16)^{16}$ which is approximately 1.53×10^{-7}. Clearly the method is not perfect since the probability of one particular character value (5) is very slightly larger than the rest. To exploit this flaw, even if the plaintext stream had one very highly probable character, the cryptanalyst would need to collect a very large sample of text. For most purposes the method is adequate.

This slight flaw could be removed, at some expense. Instead of the default condition when all sixteen tests fail, the DES output could be enciphered again to

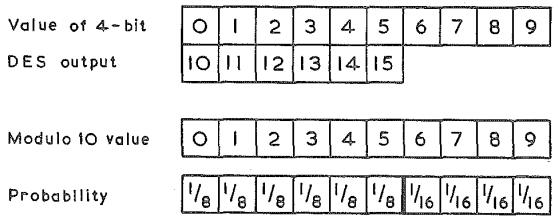


Fig. 4.10 The modulo 10 value — an uneven distribution

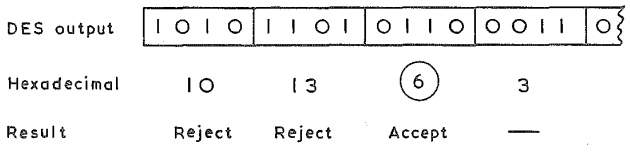


Fig. 4.11 An improved rule for modulo 10

generate a new pseudo-random block of 64 bits and the tests could begin again from the least significant end. If the second block fails, encipherment can be used to generate a third, and so on. In practice, these multiple encipherments will be rare. The result of this process would be, as far as we know, a set which is equiprobable in the range 0, 1, ... 9. But in some applications the extra encipherment cannot be allowed because of the extra time it would take. Whereas the sixteen tests can be made rapidly, some systems could not face the possibility of an indefinitely long process to produce a satisfactory character, however rare that event might be. For most practical purposes, sixteen tests followed by a default procedure is adequate.

The probability of the default condition should be checked to make sure that it does not weaken the system. Take another example, which is the 96-bit ISO graphic set. Since this is a 7-bit code, there will be nine useful blocks in the DES output and the probability of a default condition is $(32/128)^9$ which is approximately 3.8×10^{-6}.

4.4. OUTPUT FEEDBACK

Of the three methods of operation that have been described, ECB, the 'native mode', is for limited use and the two others CBC and CFB are general-purpose methods. The fourth, output feedback, is intended for applications in which the error-extension properties of the two general-purpose methods are troublesome.

Coded speech or video channels can withstand a small amount of random noise in transmission or storage, basically because they are highly redundant. The listener or viewer is not bothered by occasional clicks or spots due to faulty speech samples or picture elements. If we required it, an automatic process could detect and correct these errors, in the same way that blemishes in old films can be corrected electronically. We do not want these occasional, isolated errors to be magnified by the error-extension properties of a cipher method. For example a standard speech channel with pulse code modulation (PCM) uses 8-bit speech samples. A single error would be extended by encipherment to damage about eight further samples. This could make a bad noise in the channel or, with multiplexing, cause clicks in eight other channels.

Output feedback uses a cipher of the kind invented by Vernam; only the pseudo-random source is new. In one respect it resembles CFB operation because it can be applied to streams of m-bit characters for any value of m between 1 and 64. It has the property of all additive stream ciphers (Vernam type) that errors in the ciphertext are simply transferred to corresponding bits of the plaintext output.

Figure 4.12 shows OFB encipherment, working with m-bit characters and an m-bit wide feedback path. It resembles CFB operation in all respects except the place from which the feedback is taken. The pseudo-random stream is the result of feedback applied to the DES cipher.

The non-error-extension property of additive stream encipherment carries disadvantages with it. An active attack on the channel can produce controlled changes to the plaintext. Since OFB systems usually run continuously on a synchronous channel, the enemy may have no knowledge of where each message starts and ends, so the usefulness of this trick to the enemy might be limited.

Synchronization of the pseudo-random number generators at the two ends of

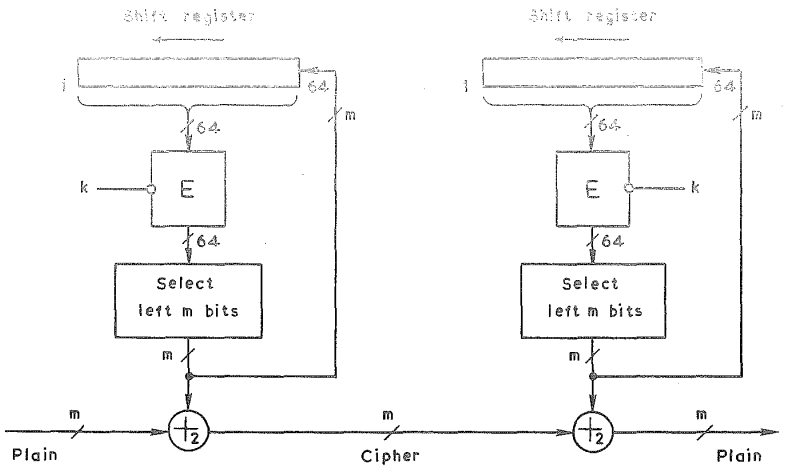


Fig. 4.12 m-bit output feedback

the line is critical. If they get out of step the 'plaintext' output at the receiver will be random. Whereas CFB operation can recover when character synchronization is restored, OFB will not if characters have been gained or lost. A system employing OFB must have a method of re-synchronization after such a failure. This is the same as restarting with a new IV.

To start up or resynchronize OFB, the shift registers must be loaded with identical values. The IV can be sent in clear form because knowledge of it does not help the enemy to construct the pseudo-random stream. In the same way as for CFB operation, the full 64-bit field need not be used. The IV sent over the line can be padded on the left with zeros to fill the shift register. There is the same trade off between IV overhead and security as in CFB initialization.

Key stream repetition

The pseudo-random stream which is added (modulo 2) to encipher and decipher is called the *key stream*. If ever the key stream repeats it weakens the cipher. If X_i is the key stream and it is used for two different plaintexts A_i and B_i, the two ciphertexts are: A_i+X_i and B_i+X_i (modulo 2 addition). Adding these, modulo 2, produces A_i+B_i, eliminating the key stream. The result is that which would have been obtained by enciphering A_i with B_i as a 'running key' cipher, or vice versa. Solution of such ciphers is sometimes easy. The enemy may not know, at first, where the key-stream repetitions occur, but if the messages are in natural language there are statistical tests which distinguish their sum from the random stream caused by X_i. Digitally coded voice and video have even stronger statistical properties. Sliding the enciphered messages and testing their sum will detect the key repetition. Such repetitions must be avoided.

Output feedback, like any other pseudo-random generator, repeats its output after a certain time, that is, it enters a cycle. The key-stream generator can be

regarded as a finite state machine with 2^{64} states, each state corresponding to one value x in the shift register of Figure 4.12. More than 2^{64} states per cycle are impossible.

Output feedback with $m=64$ iterates the function $x_{i+1}=Ek(x_i)$, which is a one-to-one mapping of x on to y. It permutes the 2^{64} values of the state variable x. When such a generator re-cycles, it returns to its initial state. Permutations can be represented by the cycles they produce and it is well known that when all the $n!$ permutations are written out as cycles, the total length of all the cycles of any length c is exactly $n!$, whatever the value of c. In other words, starting from an arbitrary state x_0, the length of cycle is equally likely to take any one of the values 0, 1, 2 ... n. In the case of OFB, where $n=2^{64}$, the mean cycle length (averaging over all keys and starting values — or with randomly chosen starting values) is $2^{63}+\frac{1}{2}$. This result depends on a significant assumption, which is that the encipherment functions $Ek(x)$ are effectively random choices among the $(2^{64})!$ permutations.

A cycle length which is random and uniformly distributed is not as safe as a cycle of known length, but at least the average length is high and the probability of a very low cycle length is small. OFB with $m=64$ is satisfactory.

The analysis of OFB with other values of m is more difficult. The function $x_{i+1}=f(x_i)$ relating each state x_{i+1} to the previous state x_i is not one-to-one. A simplifying assumption can be made, according to which it is a *random* function so that, until the cycle repeats, each new x value is effectively chosen at random. A cycle begins when any x value repeats one that has already occurred. Although this is not a good description of the OFB operation, it has been found experimentally to give the correct cycle behaviour.[5] The problem of finding cycle length is similar to the solution of the famous 'Birthday problem' which is of such importance in the analysis of data security that it is described in an appendix to chapter 9, beginning on page 279. The solution of that problem indicates that coincidence of two x values is likely to happen after about $\sqrt{n}$ iterations, where $n=2^{64}$ is the number of values from which the x values are randomly chosen. A more careful analysis described in reference 5 shows that the average cycle length for a random function is $(\pi n/8)^{1/2}$, which is close to 2^{31}. The OFB function is, however, not random and by a closer analysis it was possible to estimate the way that the average depends on m.

The largest effect of non-randomness occurs for $m=1$ and $m=63$ and it increases the average cycle length by a factor $\sqrt{2}$. The effect decreases for $m = 2, 3 \ldots$ and $m = 62, 61 \ldots$ so that the average $(\pi n/8)^{1/2}$ is accurate for most values of m. The average cycle length is always in the range 2^{31} to 2^{32}.

An average cycle length of this magnitude is not satisfactory in an encipherment scheme intended for general use. For example, a digitized speech channel at 2^{16} bit/s rate would recycle on average after 2^{16} s or 18 hours. There is a further reason why a random function is not suitable for generating a key stream. The average number of distinct cycles for such a function is very small, namely $\frac{1}{2}\ln n$, which for 2^{64} system states is only 22. The conclusion from these calculations is that OFB should be used only with full 64-bit feedback. The other feedback possibilities for OFB in Federal Information Processing Standard (FIPS) publication 81 should be avoided. The effect of this on standards is described in chapter 11 (page 349).

4.5. STANDARD AND NON-STANDARD METHODS OF OPERATION

We can now compare the properties of the four 'standard' methods of operation.

The electronic codebook method (the simple use of a block cipher) is not recommended for messages larger than one block because of the danger of 'codebook' analysis of repeated block values and the possibility of reassembling messages from known blocks. When ECB is used to encipher a block of data, the way it is used should prevent repetitions or make them very unlikely.

The most usual application of the ECB method is to encipher a key for transmission. Since a key contains 56 bits of random data, the building up of a codebook is impractical. Of course, the danger of a replay of old keys must also be considered when the security is analysed.

If very short messages (such as acknowledgements) have to be sent which fit into one block, codebook analysis should be defeated by making the block sufficiently variable. There are several ways in which the variability can be achieved. One is to add an initializing variable, effectively using cipher block chaining. Instead of sending an IV, which must be protected from manipulation, the added variable can be a serial number, incremented for each transmission. If the message is short enough, the serial number can be incorporated in it, which is more convenient for checking and resynchronizing.

An alternative is to put a date and time in the message which can be checked to detect a replay. Since a compact form of date and time, down to the second, occupies 48 bits, there will be space for 16 bits of message. Another method is to include with the message a random number. This prevents codebook analysis but does not easily guard against replays. Methods like these can be 'tailor-made' for a particular application. Compared with general-purpose protocols these methods can be more efficient and justify the special software that has to be written. In most circumstances it is better to use general methods based on CBC or CFB operation and accept their overheads.

Cipher block chaining is the recommended method for messages of more than one block. This method avoids codebook analysis generally but not at the start of the chain. Communication systems generally use chain formats which begin with a serial number so that the first block differs for all chains using a given key. In other applications the IV may be varied.

This method extends single bit errors in the ciphertext to affect two successive blocks at the plaintext output. An enemy with control of the ciphertext can introduce controlled errors into the second of these blocks, therefore the method of error recovery should not trust chains in which errors are detected.

Cipher feedback is recommended for enciphering streams of characters when the characters must be treated individually. Like CBC it operates on chains — in this case chains of characters. The problem of codebook analysis at the starts of chains is similar but in communication by CFB operation the variability of the start of the chain is usually provided by varying the IV. Error extension is present also in CFB but the security implications are different because it is the first character of the error pattern in the plaintext output which is under the control of the enemy, if he can introduce ciphertext errors. The chain format should not allow important information to appear in the last character. This requirement is

met, for example, if the last character is part of the sum check and cannot usefully be altered by the enemy.

Both CBC and CFB methods recover from bit errors, which is another way of saying that their error extension is finite. Loss of block synchronism in CBC or byte synchronism in CFB causes an indefinite garble. The case of 1-bit CFB is an exception, with no synchronization requirement beyond bit synchronism. It is often regarded, in consequence, as the most 'transparent' of the methods. Fortunately, cipher feedback is often applied with asynchronous, start-stop operation which restores the synchronism of characters immediately. Cipher block chaining is often applied to data which is in storage or is transmitted by a link procedure, such as HDLC, which restores synchronism at least in the next frame after a synchronization error. So the normal error recovery procedures can deal with the problem.

Output feedback is needed when error extension is undesirable. In this method of operation, synchronization errors are not recovered. Therefore loss of synchronism must be detected and notified in a signal back to the sender which causes a new IV to be sent, in order to restore synchronism. As an alternative where a return message is impossible, synchronization can be re-established so frequently that the loss due to an error is very brief.

The four methods of operation are versatile enough for nearly all applications. In order to guard against active threats the designer has to choose carefully the methods of forming the IV and transmitting it. He may also need to provide variability at the beginning of a chain by means of a serial number, a time and date or a random number. The four modes of operation are a set of tools, extending the versatility of the block encipherment algorithm but still requiring to be applied with some thought.

These methods do not exhaust the possibilities. We can construct other methods for using the block cipher on chains of blocks. For an example, Figure 4.13 shows how, operating on 64-bit blocks, encipherment can be incorporated into both the feedforward and the feedback paths. The symmetry argument helps us to generate such schemes. This one can be expressed by the equations

encipherment:

$$C_n = Ek_1[P_n +_2 Ek_2(C_{n-1})]$$

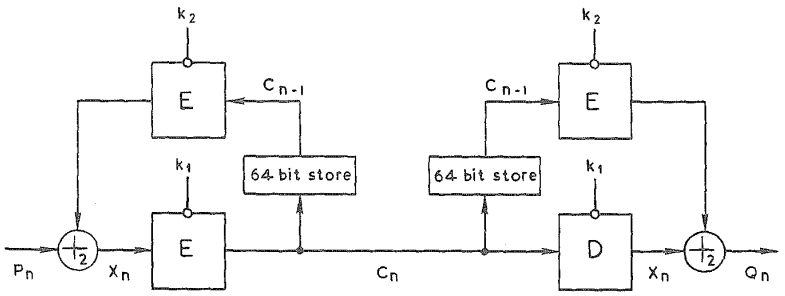


Fig. 4.13 A non-standard chaining method

decipherment:

$$Q_{n-1} = Dk_1(C_n) +_2 Ek_2(C_{n-1})$$
$$= P_n.$$

In error performance, synchronization etc., this method is like CBC but it uses two keys k_1 and k_2. To some extent these provide extra security, rather like the effect of double encipherment. We claim no special virtues for this method.

4.6. SECURITY SERVICES IN OPEN SYSTEMS INTERCONNECTION

The International Standards Organization has produced an architectural model for networking which is defined in ISO 7498 entitled 'Open Systems Interconnection — Basic Reference Model'. This OSI model is the framework for all dependent ISO standards. Unfortunately, it is not universal because the IBM *systems network architecture* (SNA) is the basis for many of today's networks. Among other computer systems suppliers, OSI is gradually achieving a dominant status. Though there is a general similarity in the approach taken by SNA and OSI, there are basic problems in interworking between these two systems. In this book we will follow the development of a systematic architecture for security services in OSI. The model is contained in part 2 of ISO 7498.[6]

Open systems interconnection is structured in the well-known seven layers from the physical layer at the bottom to the application layer at the top. The lower three layers, physical, data link and network form the communications structure and their standards have been put forward initially as CCITT recommendations. The higher layers, transport, session, presentation and application employ the communications facilities of the network layer. These upper layers can be considered as operating from end to end through the communications system. In principle, security features could be built into any layer of the OSI structure but the security architecture is more specific than this and defines, for each security service, those particular layers which can provide it.

The OSI security architecture makes a clear distinction between security *services* and security *mechanisms*. For example, encipherment is a *mechanism* which can be used as part of a number of services, of which the most obvious is that of confidentiality. The emphasis in the OSI document is on the services rather than the mechanisms, since the latter form part of dependent standards detailing how these services can be provided at the various layers of the architecture.

Definition of security services

The security services fall into five groups: confidentiality, authentication, integrity, non-repudiation and access control. Some of these can be broken down into sub-types according to the method of communication and the details of the service required. Definitions of the terms can be found in the Glossary starting on page 360.

Data confidentiality ensures that information is not made available or disclosed to unauthorized individuals, entities or processes. It can be applied either to the whole of the message or to specific fields within a message, which is called *selec-*

tive field confidentiality. Data confidentiality can also be distinguished according to whether the communication protocols used are connectionless or employ established connections between peer entities.

The word *authentication* is used in OSI in a specialized sense which is different from the way it has been used in the past. Authentication in OSI refers to the certainty that the data received comes from the supposed origin. In the context of connectionless service the term used is *data origin authentication* and is defined as 'corroboration that the source of data received is as claimed'. Where an association has been set up between peer entities the meaning of authentication can be strengthened and is contained in the term *peer-entity authentication* meaning 'corroboration that a peer-entity in an association is the one claimed'. The term authentication is not therefore extended to include the integrity of the data which are being transmitted.

Data integrity is treated as a separate concept, meaning the property that data have not been altered or destroyed in an unauthorized manner. Outside the context of OSI, this form of data integrity is sometimes called authentication but the OSI distinction can be useful and will be maintained in the remainder of this chapter.

In practice data integrity and data authentication are linked in two ways, they usually employ the same mechanisms and they are usually both required together.

Concerning the need for both services, if it can be assured that the data received come from the claimed source, this is not useful in practice if the data may have been changed in an unauthorized way. Conversely, if we know the data have been transmitted without any unauthorized changes, this is not useful unless we know that the data came from the claimed source and not from an imposter. Therefore, in practice, data integrity and authentication are required together rather than separately.

The mechanisms by which these services are obtained are the subject of chapter 5 and depend on cryptographic processes using a secret key shared between the communicating parties. In many cases, the possession of this key establishes the authenticity of the data source and at the same time allows the integrity of the data to be checked. Not all authentication mechanisms employ cryptography but the more secure forms of authentication always do.

Data integrity services can vary according to whether the communication is connectionless or employs a peer-entity association. Where an association exists the data integrity mechanism can be provided with a form of recovery mechanism. Data integrity services can be applied to the whole of the message or to selective fields, as was the case for confidentiality.

Repudiation is defined as 'the denial by one of the entities involved in a communication of having participated in all or part of the communication'. Non-repudiation is a service which can be provided in the OSI architecture in two forms, either with proof of origin or with proof of delivery. The concept of non-repudiation will be understood better in the context of digital signatures, which are the subject of chapter 9. Digital signature is one of the mechanisms by which they can be provided, the other is notarization by a trusted third party.

In the case of non-repudiation with proof of origin, the recipient of data is provided with proof of the origin of the data which will protect against any attempt by the sender to falsely deny sending the data or to deny its contents. A digital

signature can provide this service in such a way that the message and signature can be taken to a third party who can verify for himself the origin of the data. Where notarization is employed, the authority of the notarization service must be called upon to provide this proof. Non-repudiation with proof of delivery is a similar service to protect against any subsequent attempt by the recipient to falsely deny receiving the data or to deny its contents. Essentially this requires the recipient to provide a signed receipt for the message or else to send that receipt to a trusted notarization authority.

The final category of security services recognized in OSI is *access control* and this can be applied either near the source of a communication, at an intermediate point or near the destination. Access control may be needed in order to protect a network against deliberate saturation by an opponent, but normally it protects an information service against unauthorized access and is therefore frequently used at the application layer or above. Within the network layer, an access control service can provide protection for a sub-network, allowing only authorized entities to establish communication within it. The transport layer can provide a similar service.

A special form of confidentiality is *traffic flow confidentiality* which protects against traffic analysis, namely the inference of information from the observation of traffic flows. It is difficult to obtain complete protection against traffic analysis and a number of mechanisms may be required simultaneously, such as encipherment at the physical layer together with traffic padding to confuse observation of traffic levels.

The mechanisms employed to provide these services include encipherment with both symmetric and asymmetric algorithms, digital signature, integrity and authentication mechanisms, traffic padding, routing control and notarization. There are also *pervasive mechanisms* which can be applied at all levels of the OSI architecture and three of these are distinguished. The first is *trusted functionality* which means that the parts of the system which carry out security functions must be both protected and trusted. The second is called *event detection and handling* and includes the detection of attempts to break into the system and the way in which this information is acted upon. The third category of pervasive mechanism is the *security audit trail* which provides a record of what has happened and enables the security of the network to be reviewed and action taken when trouble is found.

Security services in relation to the OSI layers

Many of the services envisaged in the OSI architecture can be provided as alternatives at different layers. Table 4.1 shows, for each of the defined services, in which layers it may be provided. The session layer provides no security services so this has been omitted from the table. Many of the services provided by the application layer will actually employ security mechanisms within the presentation layer. For example, if the presentation layer changes the coding or format of some messages this can only be done prior to encipherment; therefore encipherment to provide confidentiality at the application layer may, in practice, be carried out by the presentation layer after the recoding or reformatting has been done. In this table we have combined the two forms of authentication, peer-entity

Table 4.1 Location of security services in OSI layers

	Layer of OSI					
	1	2	3	4	6	7
Confidentiality						
Connection	Y	Y	Y	Y	Y	Y
Connectionless	—	Y	Y	Y	Y	Y
Selective field	—	—	—	—	Y	Y
Traffic flow confidentiality	Y	—	Y	—	—	Y
Authentication (both kinds)	—	—	Y	Y	—	Y
Integrity						
Without recovery	—	—	Y	Y	—	Y
With recovery	—	—	—	Y	—	Y
Selective field	—	—	—	—	—	Y
Non-repudiation (both kinds)	—	—	—	—	—	Y
Access control	—	—	Y	Y	—	Y

OSI layers: 1, Physical; 2, Data link; 3, Network; 4, Transport; 6, Presentation; 7, Application.

and data origin authentication, and the two forms of non-repudiation service, with proof of origin and with proof of delivery.

The letter Y in the table means that the service should be incorporated in the standards for the layer so that the provider of this layer can include the security services as an option. Where there is no Y the service cannot be provided.

For each of the OSI layers, work is proceeding to incorporate the security services as an option. Some of the progress to date is described in chapter 11.

4.7. THE PLACE OF ENCIPHERMENT IN NETWORK ARCHITECTURE

Even at a casual glance, a computer network can be seen to have a complex structure. It consists of switching centres, often called nodes, connected by communication links and joined to multiplexers or concentrators which provide the paths to the network's host computers and terminals. In such a structure, encipherment could be employed in many different ways. A more careful study of network structure reveals that the complexity goes beyond its communication system. It has further levels of structure incorporated in the host computers and the terminals.

When encipherment is used in a network, the security properties it gives are dependent on the layer at which it is applied. The OSI model shows that potentially it can be applied at any of six layers and the effect is has depends on the network structure and on the responsibility for operating different parts of the network. For example, in a private network the nodes may sometimes be regarded as trusted facilities whereas they would not be trusted in the shared use of a public network.

The OSI view of network structure encompasses both the communications system and the intercommunicating computers, terminals etc. For the present, we take a less-detailed view and look at the main alternatives for using encipher-

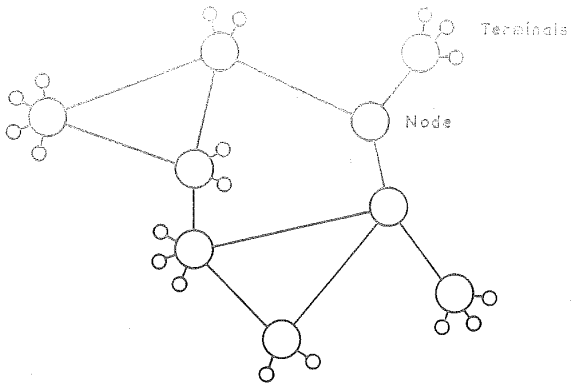


Fig. 4.14 Network of nodes — switches, multiplexors, etc.

ment in the communications part of a network. For this purpose we shall describe three ways in which encipherment can be employed, called, for the present purpose 'link-by-link', 'end-to-end' and 'node-by-node'.

Figure 4.14 shows a network as a set of switching centres or 'nodes' connected by communication lines.

The simplest place to apply encipherment is on the lines between the nodes. These communication lines are the most vulnerable part of the system. This is called encipherment 'link-by-link'. Since the lines are used for carrying bits or characters from one node to another, the encipherment in principle need take no account of the structure of the messages, such as the presence of headers. Therefore some forms of this encipherment are very simple. Information must be deciphered as it enters the nodes, because these nodes look inside the structure of the messages and must see the information in clear form. The information is completely vulnerable to attack while it is in the node: in the OSI model this form of encipherment is in layers 1 or 2.

Many users will require a different form of encipherment, which allows the content of their data to remain enciphered whenever it is in the network, even in the nodes. This can be achieved with 'end-to-end encipherment' in which the communication network is regarded as a single entity which is transparent to the enciphered message: in the OSI model this is encipherment in any of layers 3 to 7. The message content is enciphered at the point of entry to the network and deciphered at the point of exit. This makes the data as secure inside the nodes as elsewhere in the network.

We shall also consider another method of making data secure inside the nodes which is called 'node-by-node encipherment'.

Link-by-link encipherment

The line is a path of communication betwen nodes and uses a protocol or procedure which operates over this line. To be more precise, both the *physical* and *data link* layers of the architecture may be involved. Figure 4.15 illustrates link-by-link encipherment.

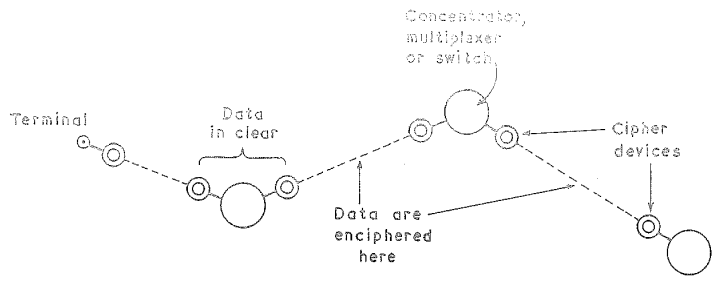


Fig. 4.15 Link-by-link encipherment

If we encipher at the lowest level, treating what passes over the link as a sequence of characters or bits, then an enemy who taps the line will see nothing of the structure of the information. He will therefore be unaware of the sources and destinations of messages and may even be unaware whether messages are passing at all. This provides 'traffic-flow security' whch means that we conceal from the enemy, not only the information, but also the knowledge of where the information is flowing and how much is flowing. By using the lowest level of encipherment, traffic-flow security can be obtained.

Consider a synchronous communication line, carrying a stream of bits. This stream can be enciphered using 1-bit cipher feedback. After the initialization, the process can continue to run as long as the line is open, recovering automatically from bit or synchronization errors. An enemy tapping the line will see only a random sequence of bits, which does not reveal whether data messages are passing or not. The designer might have to consider whether the 'infill' between actual messages should be a repetitive pattern or random. Having no knowledge of the beginning or end of messages, the enemy could not attempt any useful active attack. Supposing that he did change some of the information, this would appear as unpredictable changes in the plaintext. Detection of these changes at the lowest level of protocol would be difficult, but they should be detectable at a higher level if the error control mechanism is well chosen.

The situation is less clear when asynchronous, start-stop communication is used. In this case, also, the entire message content can be enciphered by 1-bit cipher feedback or by character-wide cipher feedback, leaving the start and stop elements unchanged. There remains one clue for the enemy, which is the rate at which characters are passing. If this traffic information must be concealed, the system requires an extra provision which sends dummy messages during quiet periods. These dummies can be generated and absorbed within nodes. In packet switching networks a logical channel could be devoted to this purpose and made to fill up a part of the unused capacity of the line.

What we have described belongs in the *physical* layer of the architecture. The *data link* layer protocols, such as HDLC or BSC, define 'frames' which pass between the nodes. If the encipherment procedure is applied to the data content of these frames, some more traffic information is given to an enemy tapping the line. This illustrates that raising the level at which encipherment is carried reduces the traffic-flow security. More details of the precise methods used for encipherment at layer 1, the physical layer, are contained in chapter 11.

Key distribution for encipherment at these low levels is straightforward. The exchange of the keys concerns only the two nodes connected by the line and is a local matter. The key can be changed at local discretion without affecting the rest of the network.

The limitations of this form of encipherment appear when we look at the higher-level functions of the network. The information contained in the nodes (the equipment, other than the lines or the link protocols) is in clear form. This could be tolerated only in a private network in which the degree of care in physical protection, personnel selection, maintenance etc. is just as high in the nodes as it is at the terminals or in the operation of host computers. Network errors, accidental program bugs or deliberate bugs introduced by an enemy might redirect a message to the wrong destination. If all the users of the network trust each other and have the same level of security clearance, this need not matter. However, even in a private network, information owned by one department may have to be withheld from another, so the kind of accident which might redirect messages should be avoided. In a public network, few users could accept the possibility of redirection of messages or the presence of clear messages in the switching or concentrator nodes.

To summarize, the use of encipherment at a low level has the advantage of providing traffic-flow security but the great weakness of protecting data only on the communication lines, and not providing adequate separation between one network user and another.

We therefore argue that the use of link-by-link encipherment should exploit its advantages and the weaknesses should be dealt with by encipherment at a higher level. When link-by-link encipherment is used in this way it should preferably be done at the physical layer, that is, applied to the individual bits or characters and without regard to the structure imposed by the link protocol. The best kind of low-level encipherment is at the lowest level, when it is supplemented by encipherment at a higher level.

End-to-end encipherment

Figure 4.16 depicts end-to-end encipherment, with an encipherment device interposed between each terminal and the network. The encipherment process must take into account the protocol of communication between the terminal and the network. The purpose of a call from one terminal to another is to set up a

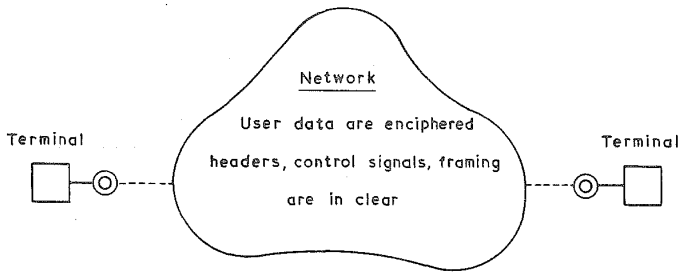


Fig. 4.16 End-to-end encipherment

circuit or virtual circuit for communicating data. Some of the data passing over the line between terminal and network are for use by the network, not the terminal, and these data cannot be enciphered because the network does not possess the key. Therefore, the encipherment device must distinguish between the 'user data', which are destined for the distant terminal, and all the rest.

To give an example from packet switching, we can see how this relates to the use of an X.25 interface. The lowest layer of the X.25 interface is the physical layer with, immediately above it, the data link layer that provides a HDLC mechanism for communicating packets. None of this changes when end-to-end encipherment is used. All that happens is that these mechanisms carry some enciphered data. At layer 3, in which packets are formed, there are about twenty-six different packet types. Most of these are concerned with call set up and clearing calls, with flow control, reset and restart. None of these packets is affected when enciphered data is handled. Only the two packet types that carry users' data need be affected, the DCE data and DTE data packets. In these, the header includes a logical channel number and the send and receive sequence numbers, forming part of the mechanism operated by the network itself to separate one virtual circuit from another and to control flow. This header must remain in clear. The user data which follows it is enciphered. The DTE and DCE interrupt packets also contain user data which is not examined by the network so that this information can be enciphered if the user wishes.

If encipherment is carried out by a unit interposed between the terminal and the network, as Figure 4.16 suggests, this unit must interpret some part of the X.25 protocol in order to find these two kinds of users' data fields, which are the subject of encipherment. In practice, encipherment might be carried out on the data before they enter the X.25 mechanism of the terminal, so avoiding much of the complexity. It is also likely that the various virtual calls carried by a single X.25 interface would have different keys, since they probably concern different users.

In order to specify exactly how encipherment of data over a virtual circuit would take place, additional protocol features are needed, if only to let the destination terminal know that a call is enciphered. In practice the protocol features associated with security are moderately complex. They can be provided either by a special security protocol or as an amendment or enhancement of one of the higher protocol layers.

All the operations of the network which set up a virtual circuit are unprotected by encipherment. The 'call request' and 'call accepted' packets can be observed passing through the network and so the progress of each call and the quantity of information carried can be monitored. There is no traffic-flow security in end-to-end encipherment. The best overall approach is to use encipherment twice, at a high level and again at a low level, because the virtues of high- and low-level encipherment are complementary. End-to-end encipherment protects the users' data everywhere in the network and link-by-link encipherment protects traffic flow information on the lines.

It might seem unreasonable to ask for traffic-flow information to be protected within the nodes, because this is where routing decisions are made, and they require traffic information to be exposed. But there are ways in which even this degree of protection can be achieved. It is done by providing in the network secure

'rerouting centres' which share encipherment keys with the users. The message hops from one re-routing centre to another, eventually finding the right destination. There can also be some dummy traffic created by the rerouting centres themselves, to conceal the flow. Such a scheme[7] requires the users to trust the rerouting centres with whom they share the keys necessary to determine the successive destinations of the messages. This scheme has been studied theoretically but seems likely to have only very specialized use, when extreme traffic security is the prime consideration.

The key distribution problem for end-to-end encipherment

The encipherment and decipherment functions at both of the terminals in Figure 4.16 must have the same key. Therefore each pair of terminals that wishes to exchange messages must somehow obtain a key for that purpose. If there are N terminals, potentially there will be $N(N-1)/2$ possible keys. In chapter 6 we shall see how session keys can be distributed using only N master keys, so the problem is not as bad as it seems. Nevertheless, key distribution for end-to-end encipherment has a degree of complexity which was not present in line encipherment, where the keys for each line were the concern of only the two nodes at the ends. The multiplicity of keys is a consequence of end-to-end encipherment's big advantage — it separates each call from all the others. If a packet goes astray it arrives at a place where it cannot be deciphered and little harm is done. We shall return to the key distribution problem in chapter 6.

Node-by-node encipherment

A hybrid method of enciphering in networks has been described under the name 'node-by-node' encipherment.[8] This shares with link-by-link encipherment the property that each key belongs to a particular line, so the key-distribution problem is made easier. This has the corresponding effect that an error in the system could deliver messages to the wrong destination. But if ease of key distribution is an overriding factor, node-by-node encipherment might be attractive. It is used in some EFT networks.

In this form of encipherment (unlike link-by-link encipherment) only the users' data are enciphered, not the headers. This enables the switches, multiplexers or concentrators to operate on the messages as though they were in clear form, but without having access to the users' data.

There remains the problem of changing the encipherment from one key to another as data come in on one line and go out on another. This function is carried out in a physically secure module which is contained in the node but given special protection. The secure module contains all the keys which the node must use, one for each of its lines. Figure 4.17 shows the principle. In order to carry out the re-encipherment the secure module must be told the identity of the input and output lines, after the node has made its routing decision. With these data, the module extracts the necessary keys, deciphers the data stream with the incoming line's key and re-enciphers it with the outgoing line's key. Clear users' data never exists in the less-protected part of the node and the risk associated with line encipherment is avoided.

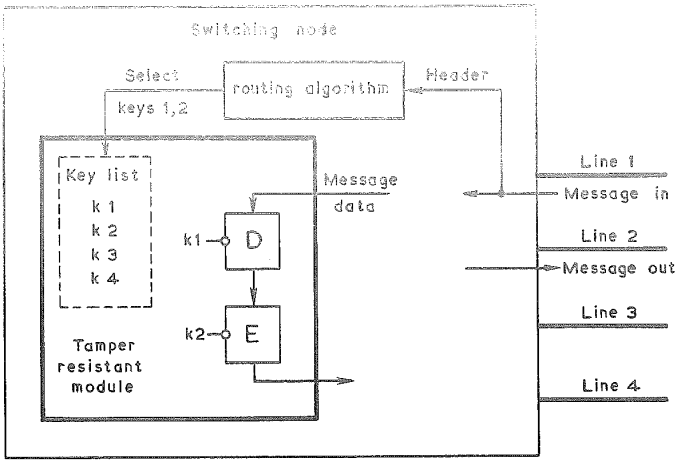


Fig. 4.17 Node-by-node encipherment

But in removing one of the risks of link-by-link encipherment, we have lost its main virtue. All the headers and routing data are sent through the network in clear. Therefore node-by-node encipherment achieves very little since it has neither the main virtue of link-by-link encipherment, which is traffic security, nor that of end-to-end encipherment, which is separation of the users' data.

A best place for encipherment in network architecture?

A rule of thumb could be formed from the preceding arguments, saying that encipherment is most useful at either a very low or a high level in network architecture. The low level, possibly in or just above the physical layer, provides traffic-flow concealment. The higher level, being one of the end-to-end protocol layers, separately protects data flows. But there are other possibilities. For example if link-by-link protection against active attack is needed encipherment at layer 2 may be preferred. For many commercial purposes, traffic flow security is not important and an end-to-end encipherment method will be all that is required.

The backbone of data security will remain end-to-end encipherment, in spite of the key-distribution problem.

REFERENCES

1. *DES modes of operation*, Federal Information Processing Standards Publication 81, National Bureau of Standards, US Department of Commerce, December 1980.
2. *Information processing — modes of operation for a 64-bit block cipher algorithm*, International Standard ISO 8372, International Organization for Standardization, Geneva, 1987.
3. *7-bit coded character set for information processing interchange*, International Standard ISO 646, International Organization for Standardization, Geneva, 1973.
4. *Guidelines for implementing and using the NBS data encryption standard*, Federal Information Processing Standards Publication 74, National Bureau of Standards, US Department of Commerce, April 1981.

5. Davies, D.W. and Parkin, G.I. 'The average cycle size of the keystream in output feedback encipherment', *Lecture Notes in Computer Science* No. 149, Cryptography, (Proceedings, Burg Feuerstein 1982), Springer Verlag, 263–279, 1983.
6. *Information processing — OSI Reference model — Part 2: Security architecture*, International Standard ISO 7498/2, International Organization for Standardization, Geneva, 1988.
7. Chaum, D.L. 'Untraceable electronic mail, return addresses and digital pseudonyms', *Comm. ACM*, **24**, No. 2, 84–88, February 1981.
8. Campbell, C.M. 'Design and specification of cryptographic capabilities', *Commun. Soc. Magazine*, **16**, No. 6, 15–19, November 1978.

eam in output feed-
Cryptography, (Pro-
3.
tecture, International
ation, Geneva, 1988.
ligital pseudonyms',

ilities', *Commun. Soc.*

Chapter 5 AUTHENTICATION AND INTEGRITY

5.1. INTRODUCTION

A piece of data, a message, file or document is said to be 'authentic' when it is genuine and came from its reputed source. In a broader sense we may also mean that the source had the authority to issue it. We might ask whether a payment is authentic in the sense that it corresponds to a genuine invoice. The concept of authentication is therefore very wide. For the purpose of this chapter we shall limit the concept to the following kinds of situation.

1. A message passes from A to B through a communication network. A and B may variously be persons, procedures or 'entities' in a hierarchy of protocols. For a general term we will call them *entities*. We want B to know that the received message came from A and has not been changed since it left A, in other words that it is genuinely A's message. It is not reasonable to separate the authenticity of the message from the identity of A, if we believe there may be enemies about. A message from A that has been undetectably modified is not different in its risks from an unmodified message that seemed to come from A but did not.

2. A message that has been stored by A is later read from the store by A or by B, a different entity. We want the reader of the data to know that it was stored by A and was not changed by others before it was read. Since stored data can be changed legitimately it may be necessary for the reader to know the originator and all those entities which subsequently altered the data. We shall see that a 'version number' is sometimes needed in stored data to keep track of alterations.

3. Two entities A and B undertake a transaction in which messages pass between them, alternately from A to B and B to A. For the transaction to be authentic, A must verify the identity of B, B must verify the identity of A, and the contents of each message in the interaction must be genuine as received — each after the first must be a genuine reply to the last message sent.

Authentication is the procedure for ensuring authenticity in any of these situations. The word *authenticator* is used for the specially constructed items of data used for this purpose. The question of secrecy of information can often be separated conceptually from its authentication. Obviously encipherment protects against passive attack (eavesdropping) whereas authentication protects against active attack (falsification of data and transactions).

In the terminology of Open Systems Interconnection a distinction is made between *authentication* and *data integrity*. Authentication refers to the means by which the parties to a communication verify each others' identities. If it is a two-way

conversation with an established association, this is known as *peer-entity authentication*. For a single message which is not part of a connection the corresponding service is *data origin authentication*. The security service which prevents unauthorized modification or destruction of data is known as *data integrity*. We have already noted that authentication and data integrity are both required to make communications safe against active attack. One of these by itself provides little useful help. Furthermore, it is quite usual for the same mechanism to provide both authentication and data integrity. It is quite common to use the word *authentication* in the more general sense of providing all the protection needed against an active attack. We shall therefore adopt the usage of Open Systems Interconnection in the OSI context, but otherwise we shall use the term authentication in the more general sense, to include integrity.

Active attack is much more complex than passive since there are many different ways of altering data. Among the threats to be considered are; (a) alteration, deletion, addition; (b) changing the apparent origin; (c) changing the actual destination; (d) altering the sequence of blocks or items; (e) using previously transmitted or stored data again; and (f) falsifying an acknowledgement.

It is noteworthy that encipherment, by itself, only protects against inserting *new* data. Most of the attacks listed above can be carried out, in principle, using old data that is available already in enciphered form. We shall detail in section 5.7 below the extra precautions needed for full protection. Since it is difficult to protect a widespread system from enemy interference the most that can be ensured is that these active attacks are detected.

In this chapter we shall first consider methods of protecting data against accidental errors in preparation and transmission and then broaden the discussion to examine methods of authenticating data both by encipherment and by special authentication functions.

Papers and signed documents are an important part of business transactions, usually setting out instructions or promises to perform actions. Because of this reliance upon documents their integrity has assumed great importance. On the other side, criminals have found much to gain from the successful falsification of business documents.

The advent of digital communication networks is causing a revolution in commercial practice. Transactions of a type which have in the past involved exchange of paper documents are now conducted by network messages; this trend can be expected to increase until the transport of messages by data networks dominates commercial practice. The accuracy and integrity of transmitted messages are therefore of very great significance.

Recognition of a valid document has customarily relied upon the written signatures of the parties to an agreement. The signature serves at least two purposes — it indicates the assent of the signatory to the terms of the document and it vouches for the accuracy of the document as signed. Indication of assent by means of a signature is a kind of proof of origin for the document; we shall return to this in detail in chapter 9. Here we are concerned with the proof of accuracy of the document, which is one of the aspects of 'authentication'.

In the following sections we treat in turn defences against the various kinds of error, leading up to *message authentication* which aims to provide data integrity through the ability to detect deliberately introduced changes.

5.2. PROTECTION AGAINST ERRORS IN DATA PREPARATION

A first step in ensuring the integrity of stored or transmitted documents and messages is to check that they have been captured correctly at the preparation stage. There are many ways of achieving this, such as double keyboarding (preferably with different operators) with automatic comparison. An alternative method, particularly relevant in the preparation of financial documents, is to use a check digit or digits — a *check field*.

It is well known that keyboarding errors usually fall into well-defined classes, such as single characters typed wrongly, adjacent characters transposed and characters repeated or omitted. Where trailing zeros are involved the number of zeros may be wrong. A check field provides a simple function of the numerical characters in the important fields (date, amount, currency, account identity, transaction serial number etc.) of a document in such a way that it is highly probable that mistakes of these kinds will be detected.

Before the data entry (keyboard) operation begins, the data to be entered must be provided with a check field. Sometimes this is permanently attached; thus part numbers or account numbers can be constructed in this way — usually with a single check digit. After data entry the computer checks the correctness of the check field and rejects those which indicate errors. In the case of entirely new data the check field is provided by human effort, but this is not obviously better than having two independent operators enter the data. Use of a check field is not a substitute for the security provision of having at least two 'independent' people involved in data entry. An operator could maliciously enter wrong data as part of a fraud attempt, while giving the correct check field, since no secret is used in forming it. The principle of 'dual control' is applied by having one person enter the data and another check it, perhaps an inspector belonging to another department. The value of a check field in this case is that it reduces the number of genuine errors to be picked up by the inspector. It can also be argued that the presence of a large number of errors to be corrected increases the opportunity for fraudulent changes.

An excellent source of good check-field schemes is the International Organisation for Standardization (ISO) Draft Standard 7064[1] entitled *Data processing — check characters*. The kinds of error that occur in data entry have been measured, for example 80 per cent of all errors are single character errors and 6 per cent are adjacent transpositions. Check digits to detect errors in numerical fields are best calculated in modulo 11 arithmetic (since 11 is prime) and with different weights in each column. One method using modulo 11 arithmetic is in turn to add each digit, then multiply by 2. The process starts from the left with the initial value 0 and the check digit is chosen so that the final value of 'balance' is 1. Here is an example with the given number 530128; in the example the Roman symbol X is used to represent the value 10 in modulo 11 arithmetic.

		check	
digit d	5 3 0 1 2 8	4	
digit c	0 X 4 8 7 7	8	$c_i = 2y_{i-1}$
digit y	5 2 4 9 9 4	1	$y_i = c_i + d_i$.

The starting value of c is 0. The c digit is added to the given number's digit to

produce y, in the first column this is 5. To get the next c value this is multiplied by 2, giving X in our notation. Then X+3=2, modulo 11, etc. Finally, the c value 8 requires a check digit 4 to produce the 'balance' 1. The effect of this calculation is a weighted sum, modulo 11 with the following weights, starting from the rightmost digit (the check digit):

1, 2, 4, 8, 5, X, 9, 7, 3, 6, 1, 2, ... repeating.

These values are 2^i, modulo 11, for $i=0, 1, \ldots$ and it can be seen that $2^{10}=1$, so the weights repeat after 10 columns. The appendix to chapter 8 (page 245) explains the mathematical basis for methods like this.

It is very important to note that use of a check field does not provide any protection against deliberate fraud. It is quite easy to generate a fraudulent version of a document which yields the same value of check digits as the genuine version. It is equally easy to create an entirely false document and generate a corresponding value for its check field, because the method of generating the check field is not kept secret. Nevertheless the check field has a useful role to play in handling data — to reduce genuine errors.

5.3. PROTECTION AGAINST ACCIDENTAL ERRORS IN DATA TRANSMISSION

In any communication system, users need assurance that their messages are delivered with neither accidental nor deliberate alteration. Accidental alteration of a message transmitted over a noisy channel is so frequent that it must be detected, and corrected either by retransmission or by application of a forward error-correcting code.

To detect accidental errors in a communication system a range of techniques is available; use of the eighth bit for parity in a 7-bit character code is perhaps the best-known and simplest example. On a slightly more sophisticated level, 'longitudinal parity' may be applied to a message, providing check digits on groups of characters by modulo 2 addition of successive words of 8 or sometimes 16 bits. Parity checks suffer from the serious defect that they may fail to detect the occurrence of a group of errors. For this reason methods have been devised to cope with a wider range of error patterns.

Cyclic redundancy checks

The best known error detection method is the cyclic redundancy check (CRC), which operates by treating the binary patterns as polynomial forms, 'dividing' a block of data at the transmitter by a predetermined polynomial and taking the 'remainder' of this arithmetic step as the check field which is appended to the data and transmitted with it. At the receiver, the received data is subjected to the same arithmetic process and the remainder from this operation compared with that received as the transmitted check field. Agreement of the two shows that it is highly unlikely that transmission errors have occurred. Recommendation V.41 of the International Telephone and Telegraph Consultative Committee (CCITT)[2] defines a standard version of this process providing a 16-bit CRC for use on communication lines; this is applied in line protocols such as HDLC. The CCITT V.41

CRC detects all single and double bit errors, all errors with an odd number of bits, all burst errors of length 16 or less, 99.997% of 17-bit error bursts and 99.998% of 18-bit and longer bursts. An extension of CRC can be applied to forward error correction, in which certain classes of errors can not only be detected but also corrected without calling for retransmission of the faulty block. This application is somewhat cumbersome and is not often found in practice. The more powerful the degree of forward error correction, the greater the amount of data expansion, and the overheads involved may be appreciable. An interesting proposal from British Telecom[3] brings forward error correction into play after it has been discovered that a particular channel is subject to serious noise, thus avoiding unnecessary overheads when a channel is performing satisfactorily.

Though checks of this kind provide strong protection against the effects of accidental error, they are not sufficient to protect transmitted data against active attack because the method of creating the check data, including the parameters of the CRC polynomial, is public knowledge. The active line tapper has only to divert the transmitted message, make whatever alteration he desires and recalculate the contents of the check field. The reconstituted message can then be sent on its way to the intended receiver where it passes the prescribed check and, in the absence of other evidence to the contrary, would be accepted as authentic. A more effective method is required in order to detect deliberate alteration.

5.4. DATA INTEGRITY USING SECRET PARAMETERS

The weakness of the methods we have discussed so far, from the point of view of secure authentication, is that they are based on parameters which are not secret — anybody can forge the data in the check field accompanying a message. Methods are needed which depend on a secret key known to sender and receiver only, information which is denied to any third parties. The distribution of this secret key represents problems like those of encipherment keys.

Figure 5.1 shows the principle of the method. It is based on a public algorithm represented by the function A(k,M). The key k is known to sender and receiver, but not to the enemy. The message M to be authenticated may be of any length and the function A(k,M) must depend on every bit of the message. The function A is the *authenticator* that accompanies the message to its destination. The receiver

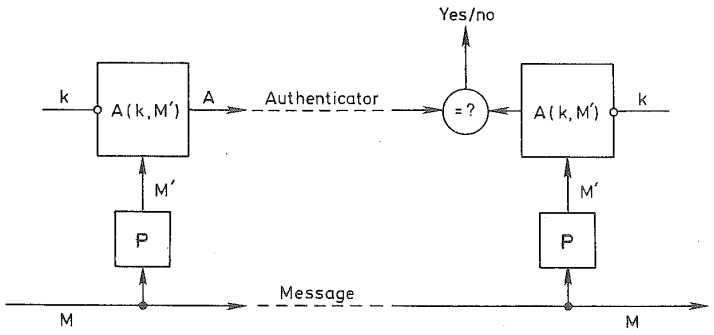


Fig. 5.1 Principle of data integrity checking

also computes A(*k*,M) and compares this with the received value A. If they are equal, the message is accepted. Sometimes this value A is known as a *MAC* or *message authentication code*. Another term is *cryptographic check value*, which is the OSI preference.

Authenticators have long been applied to financial messages sent by telex. With no processing equipment the computation must be done by hand and the nature of the algorithm, based on table lookup and addition, was matched to this requirement. In order to simplify the process, only selected fields are subjected to the algorithm. This is an example of *preprocessing* the message, shown in the figure by the function P. If a message is sent by a system that can change its coded form, the preprocessing must be done in such a way that only the essential and invariant features remain. For example a communication or storage system might treat 'new line' as equivalent to 'space' or it might remove or insert spaces or ignore the difference between upper and lower case letters. The preprocessing would then replace the 'new line' character by a space, reduce multiple spaces to one space and change lower-case letters to upper case. The message is transmitted without these changes; the altered form enters the authentication function. These operations carry a danger that significant distinctions may be obscured. Data communication networks of recent design are usually 'bit-sequence transparent', they preserve all details of the coded message — Teletex has this property for example. We recommend in these cases that there should be *no* preprocessing and that the unchanged message should enter the A(*k*,M) function.

The message conventions needed to complete the definition of authentication are, (a) the precise extent of M, where it begins and ends, and (b) the way in which the value A is formatted and located in the message as sent. A convention sometimes used is to reserve a place for A in the message format and replace it by a known constant value Q for the calculation of the authenticator. Figure 5.2 shows this method, where the fields F_1, F_2, ... F_n comprise the message content. If Q, known to the sender and receiver, is otherwise kept secret, it becomes, in effect, part of the key.

The size of the authenticator field is fixed by the function A(*k*,M) and is typically between 16 and 32 bits, which is usually much shorter than the message. It follows that many messages have the same A value and this is a potential

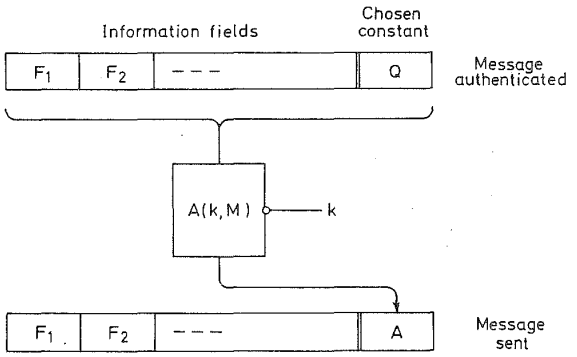


Fig. 5.2 Using a constant value, Q, in the authenticator field

weakness because the message could be changed. But without knowledge of the key the enemy does not know what other messages could be used. A good authenticator function produces pseudo-random output so the enemy is not helped by statistical knowledge, either. If the enemy relies on luck he can change the message and hope that the A value is still correct, but his probability of success is 2^{-n}, where n is the number of bits in the field. Even at 2^{-16} this is in many cases not a large enough probability to be useful. Trying enough times to have a useful chance of success would alert the receiver to a serious attack on the system. The customary size of the authenticator field is 32 bits.

An authenticator or cryptographic check value can be used on stored data in the same way as for communications. Anyone reading the data and knowing the key can check (with a very small chance of error) whether the stored data were changed by an enemy.

Like encipherment, the use of an authenticator value does not itself detect a replay attack, in which previously authenticated data are substituted, with the authenticator employing the same key. For example, during the currency of one key, data in a store may be changed by the legitimate user and the new authenticator stored. An enemy can restore the old values together with their authenticator and this interference is undetected by those legitimately reading and checking the data. The extra precautions needed to complete the defence against active attack are described later.

5.5. REQUIREMENTS OF AN AUTHENTICATOR ALGORITHM

The requirements for a good authenticator algorithm (integrity check) closely resemble those for a cipher but are somewhat easier to meet because there is no need for the inverse function to exist, as it must for decipherment. Like a cipher, the authenticator must protect its key from discovery, but this is not the full story because the function could be discovered, in principle, without knowing the key.

Cipher strength can be measured either by the 'known plaintext' or the 'chosen plaintext' criterion, and so can the strength of an authenticator. The most severe test would be to allow the enemy to submit a large set of messages and be given the corresponding A values. With this information, could he 'authenticate' a bogus message of his own? To make the criterion very strict indeed, he should not be able to find the A value for *any* message for which he has not been given it. In practice, authentication facilities are carefully guarded to prevent A values being given to unauthorized messages, so the 'chosen message' criterion is more severe than practice suggests. Also the enemy is interested in authenticator values for a specific class of bogus messages, not just any pattern of bits that suits his incomplete knowledge of the authenticator function. However, using the strict criterion is a good way to build in a large safety margin. The strict criterion may be the only one that can be defined easily, since it is impossible to characterize the set of 'good' messages for which A values will be 'supplied' to the enemy and the set of 'bogus' messages that he would like to introduce.

It is possible to describe how a good authenticator should behave, though this is not an effective criterion of strength. Changing any bit of message or key should alter about half of the bits of the A value. The full range of key values should

be effective in the sense that there are no two key values for which the two authenticator functions $A(k_1,M)$ and $A(k_2,M)$ are closely related — whatever that means. For example these functions should exhibit no correlation as M is varied. These properties should be true for all keys and key-pairs respectively. As with a cipher, it is easy to state the properties a good function should have, but there is no set of tests which can be said to complete the testing of an authenticator.

As with a cipher, one of the possible methods of attack is an exhaustive search for the key. Whenever a message with its authenticator can be read, 'known plaintext' is presented to the enemy, but there are two reasons why key searching for authenticators is harder than for ciphers.

First the authenticator value is a small number, typically between 16 and 32 bits in length. If each key value is tested and there are, say, 56 bits in the key and 24 in the authenticator, the first message tested will pass about 2^{32} key values as candidates. A second message is needed and this reduces the number to 2^8 and a third message reduces these to the one actual key.

Secondly, each test of a key involves the whole of one message. Consider, for example, a 640-bit message using cipher block chaining (CBC) with the DES algorithm and compare this with the authentication based on the same method. There are ten blocks of message. Searching for the correct key out of 2^{56} possibilities requires on average 2^{55} tests. For the enciphered message, any block will serve for the tests except the first (because the initializing variable (IV) is not known). With one block alone, because it has 64 bits, there is a high probability that only one 56-bit key will be found to fit the ciphertext. In the case of authentication, 2^{55} tests on average must be made with the first message and each of these employs DES operations. The 10×2^{55} operations make a greater task for the searcher. After this, 10×2^{31} and 10×2^{7} more operations are needed for the test with a second and a third message. Clearly all candidates passing the first test should be subjected at once to the other tests because the extra time involved is small and this enables the correct key to be identified before all 2^{56} key values have been tried. Key searching is only a 'last resort' method when other methods fail, nevertheless it is apparent that authentication is usually less vulnerable than encipherment.

The biggest practical use of authentication has been for financial messages, such as payments. For example the Society for Worldwide Interbank Financial Telecommunications (S.W.I.F.T.) system[4] (see page 289) uses an authenticator function but does not publish it. Another well-known authenticator is the 'Data Seal'[5] which is a proprietary system, also unpublished.

An ISO standard 8730[6] describes the way in which a Message Authentication Code is formed, in particular the format options for the text, field delimiters and character coding. There are optional rules for editing messages prior to authentication. ISO 8731[7] describes two approved algorithms. The first is the well-known application of DES in the cipher block chaining mode. The second is called MAA, for *Message authenticator algorithm*, described below.

The Decimal Shift and Add algorithm

A suggestion was made in 1980 by Sievi for a message authentication method. Sievi's method depends on the existence at sender and receiver of two secret ten

digit decimal numbers, b1 and b2, which form the key that controls the operation of the algorithm. The two numbers provide the starting points for two parallel computations. The message to be authenticated is regarded as a string of decimal digits. This is immediately suitable for the numerical part of payment messages and for the alphabetic part it is proposed that a pair of digits is used to encode each alphanumeric character. The message is split up into blocks of ten decimal digits; any incomplete block at the end of the message may be padded with zeros on the right to bring it up to ten decimal digits. The algorithm works entirely in decimal arithmetic and is designed for use in a programmed decimal calculator having at least ten digit capacity. The method is called Decimal Shift and Add (DSA).

The blocks of the message to be authenticated are taken one at a time and enter two parallel computations until the message blocks are exhausted; then follow ten more cycles of the algorithm using zero blocks as input. The result of this is to produce, in the parallel computations, two ten decimal outputs, z1 and z2, and these are combined together to form the message authenticator.

The basic operation of the computations is a 'shift and add', defined as follows. Let D be a ten decimal digit number and x be a single decimal digit; $R(x)D$ is the result of a cyclic right shift of D by x places.

Thus:

$$R(3)\{1\,2\,3\,4\,5\,6\,7\,8\,9\,0\} = 8\,9\,0\,1\,2\,3\,4\,5\,6\,7.$$

Then the shift and add function is defined by

$$S(x)D = R(x)D + D \qquad \text{(modulo } 10^{10}\text{)}.$$

Thus, in our example:

$$D = 1\,2\,3\,4\,5\,6\,7\,8\,9\,0$$
$$R(3)D = 8\,9\,0\,1\,2\,3\,4\,5\,6\,7.$$

Add, to produce

$$S(3)D = 0\,1\,3\,5\,8\,0\,2\,4\,5\,7.$$

Since the arithmetic is modulo 10^{10}, carries beyond the tenth position are ignored.

Let us now consider the authentication of a message of two blocks, respectively m1, (1 5 8 3 4 9 2 6 3 7) and m2, (5 2 8 3 5 8 6 9 0 0). We select for b1 and b2 the values:

$$\text{b1} = 5\,2\,3\,6\,1\,7\,9\,9\,0\,2$$

and

$$\text{b2} = 4\,8\,9\,3\,5\,2\,4\,7\,7\,1.$$

The two parallel streams of computation begin by adding m1 to b1 and b2, respectively, modulo 10^{10}.

STREAM 1		STREAM 2	
m1	1 5 8 3 4 9 2 6 3 7	m1	1 5 8 3 4 9 2 6 3 7
+b1	5 2 3 6 1 7 9 9 0 2	+b2	4 8 9 3 5 2 4 7 7 1
result p	6 8 1 9 6 7 2 5 3 9	result q	6 4 7 7 0 1 7 4 0 8.

These results are the first intermediate values, p and q, now to be subjected to the first 'shift and add' operations. For stream 1 the number of places shifted is determined by the first digit of b2, (4), and for stream 2 by the first digit of b1, (5). So we have:

$$\begin{aligned} p &= 6\,8\,1\,9\,6\,7\,2\,5\,3\,9 & q &= 6\,4\,7\,7\,0\,1\,7\,4\,0\,8 \\ R(4)p &= 2\,5\,3\,9\,6\,8\,1\,9\,6\,7 & R(5)q &= 1\,7\,4\,0\,8\,6\,4\,7\,7\,0 \\ S(4)p &= 9\,3\,5\,9\,3\,5\,4\,5\,0\,6 & S(5)q &= 8\,2\,1\,7\,8\,8\,2\,1\,7\,8. \end{aligned}$$

Let us call these values r and s, and add the next message block, m2, in each stream.

$$\begin{aligned} r &= 9\,3\,5\,9\,3\,5\,4\,5\,0\,6 & s &= 8\,2\,1\,7\,8\,8\,2\,1\,7\,8 \\ +\text{m2} &= 5\,2\,8\,3\,5\,8\,6\,9\,0\,0 & +\text{m2} &= 5\,2\,8\,3\,5\,8\,6\,9\,0\,0 \\ \text{result } u &= 4\,6\,4\,2\,9\,4\,1\,4\,0\,6 & \text{result } v &= 3\,5\,0\,1\,4\,6\,9\,0\,7\,8. \end{aligned}$$

These are the second intermediate results, u and v; there follows the second shift and add operation. For stream 1 the number of places shifted this time is determined by the second digit of b2, (8), and for stream 2 by the second digit of b1, (2). So we have

$$\begin{aligned} u &= 4\,6\,4\,2\,9\,4\,1\,4\,0\,6 & v &= 3\,5\,0\,1\,4\,6\,9\,0\,7\,8 \\ R(8)u &= 4\,2\,9\,4\,1\,4\,0\,6\,4\,6 & R(2)v &= 7\,8\,3\,5\,0\,1\,4\,6\,9\,0 \\ S(8)u &= 8\,9\,3\,7\,0\,8\,2\,0\,5\,2 & S(2)v &= 1\,3\,3\,6\,4\,8\,3\,7\,6\,8. \end{aligned}$$

At this stage, the message data having been exhausted, the construction of the authenticator should properly involve ten more cycles with zero message blocks. We omit these to shorten our description and take the two last results to represent z1 and z2. The algorithm concludes by adding z1 and z2, modulo 10^{10}

$$\begin{aligned} \text{z1} &\quad 8\,9\,3\,7\,0\,3\,2\,0\,5\,2 \\ +\text{z2} &\quad 1\,3\,3\,6\,4\,8\,3\,7\,6\,8 \\ \text{result} &\quad 0\,2\,7\,3\,5\,6\,5\,8\,2\,0. \end{aligned}$$

This authenticator has ten decimals but evidently it was considered that a six decimal value is sufficient, giving a 10^{-6} probability of a false match. To reduce the size while using all the ten digits, the number is 'folded' by having the top four digits added modulo 10^6 to the other six, thus

$$\begin{aligned} & 5\,6\,5\,8\,2\,0 \\ & \quad\;\; 0\,2\,7\,3 \\ \text{result} \quad & 5\,6\,6\,0\,9\,3. \end{aligned}$$

This final value is the message authenticator (except for the short-cut we made for ease of explanation).

Successive digits of b1 and b2 control respectively the amount of shift in streams 2 and 1 at each stage. The number of steps is always more than ten because of the added ten blocks of zeros. After each ten steps the digits of b1 and b2 are used again in succession. Thus each key value enters in two ways. It forms the starting value of one stream of calculation and controls the shift operations of the other.

The methods by which this algorithm defeats an enemy can be looked at in different ways. If all the calculation had been made with modulo 10^{10}, there

might have been an arithmetic way to analyse the result, but the cyclic shift operations do not fall into this category. They are, in fact, calculations with modulus $10^{10}-1$, since each time a digit d is moved cyclically it changes the value by $(10^{10}-1)d$. It is this combination of calculation with two moduli that complicates analysis. Another source of strength is the combination of the result of two parallel calculations. If the ten digit result were known this would give no easy way to calculate backwards. The final set of ten zero blocks helps to reduce the vulnerability of the final message block. Otherwise a single digit change of the final message block would reveal the parameters in the last shift operation, which are two of the key digits. The use of each key number in two distinct roles complicates any attempt to separate out the influence of one key digit.

The DSA algorithm is well designed to exploit the abilities of a programmed decimal calculator. It was considered as a possible international standard, but, in the event not adopted. Small calculators employing the DSA algorithm are now used in home banking both to provide a message integrity check and also for the authentication of the user to the bank. This application is described in chapter 7.

Message Authenticator Algorithm (MAA)

The ISO working group recognized that more than one type of authenticator algorithm was needed and there seemed to be a need for an algorithm suitable for programmed computers of the binary type. Though it is known that the S.W.I.F.T. and Data Seal Algorithms meet this need, they are not available for use as a published standard.

This requirement and the need expressed by some UK banking organizations led one of us and a colleague to produce a proposal for an authenticator algorithm suited particularly to binary computation. We called it a 'main frame' algorithm because it is based on 32-bit arithmetic and logical operations which are traditionally found in a main frame computer. The advancing powers of microcomputers soon changed this situation. There are both hand-held calculators and microprocessors which can conveniently implement this algorithm, but it was primarily designed to give a combination of speed and security in any 32-bit computer which can rapidly produce the double-length result of multiplying together two 32-bit binary numbers. Its design was suggested by the DSA algorithm and it employs the same general principles. It is now incorporated in the international standard ISO 8731-2 and is known as *MAA*.[7]

The 'shift and add' operation of the DSA algorithm is similar to a multiplication with modulus $10^{10}-1$ in which the multiplier takes the binary form 10 ... 01, having an adjustable number of zeros between the two 1 bits. Using an actual multiplication enables this same operation to be used for 'mixing' the digits between each introduction of a new block of message, with a less restricted multiplier value.

This multiplication scheme presents two dangers, the possibility that the result may become (and remain) zero and the property that, if the modulus has factors, any of these which appear in the intermediate result will remain there through the rest of the computation. To avoid the first of these problems the multipliers were deliberately constructed with certain bits fixed as zeros and ones, and to

	V: = CYC(V);	V and W derived from the key
	E: = XOR(V,W);	
X: = XOR(X,M_i);	Y: = XOR(Y,M_i);	M_i is a message block
F: = ADD(E,Y);	G: = ADD(E,X);	
F: = OR(F,A);	G: = OR(G,B);	A,B,C,D, are constants
F: = AND(F,C);	G: = AND(G,D,);	
X: = MUL1(X,F);	Y: = MUL2A(Y,G)	Modular multiplications, see text

Fig. 5.3 The basic loop of a 'main frame' authenticator calculation

avoid the second problem one of the parallel computations is done with modulus $2^{32}-1$ and the other with modulus $2^{32}-2$. The second of these is $2(2^{31}-1)$ and $2^{31}-1$ is a prime.

Figure 5.3 shows the basic loop of the authenticator calculation with the two streams of computation shown side by side. CYC(V) is a 1-bit cycle left shift of V, MUL1 is multiplication modulo $2^{32}-1$ and MUL2A is multiplication modulo $2^{32}-2$. A, B, C and D are constants which help to condition the multiplier values F and G. The ADD operation is modulo 2^{32} and the logical operations operate in parallel on the 32 bits. Each multiplier is derived from the other calculation stream using also the value E which is derived from V and W and changes at each step (repeating after 32 blocks).

As in the DSA algorithm, the final authenticator result is derived from both streams of computation, in this case by an XOR operation applied to the final results. A 32-bit authenticator is derived and it is left to the application to decide the size of authenticator field needed and the way in which the 32 bits will be used to generate this value.

The keys which control this authenticator are two 32-bit numbers J and K. These are used in a 'prelude' which is a complex calculation, employing multiplications with the same two moduli, and which generates six 32-bit numbers employed to control the computation. Though the prelude is a relatively complex calculation, it is used only once when new keys are introduced. The six values can then be stored and employed in any number of authenticator calculations. Two of these values are the starting values for X and Y in the two streams of computation, two of them are V and W used to derive the changing values E and two of them, S and T, are used as final blocks appended to the given message to round off the computation, this part of the procedure being known as the 'coda'.

The MAA algorithm derives its strength from the difficulty of analysis when calculations are carried out with a number of moduli. Not only are there multiplications with two different moduli, but also arithmetic operations modulo 2^{32} and the logical operation XOR. Also, the logical operations which generate the multipliers F and G are non-linear, so that any arithmetic analysis is exceptionally complex. By deriving the six quantities from the key in a complex manner any method of analysis which was capable of partly deriving one of these values would be unlikely to give a clue to the others. It is particularly important that the final blocks S and T which are appended to the message for authenticator purposes are secrets derived from the key values.

The computation time for authenticator values is proportional to the message size except for the small addition of blocks S and T. This makes it practicable to

use the algorithm on short messages as well as on long files. For very long mesages the standard specifies a special mode of operation.

Authentication methods using the standard 'modes of operation'

Wherever DES hardware is available it is convenient to use it for authentication and for this purpose 'modes of operation' have been defined in Federal Information Processing Standard (FIPS) 81 and in ISO 8731-1.[7,8] Any block cipher algorithm could be used in these modes, but in practice the DES will normally be employed. This is convenient only if a hardware DES function is available because the algorithm is slow and inconvenient in software. There are two authentication methods based respectively on the cipher feedback (CFB) and cipher block chaining (CBC) methods used for enciphering messages (see pages 81–90). CBC operates always on 64-bit blocks, but a message can be padded out with zeros to fill the last block. CFB operates on appropriate-sized characters such as 8-bit bytes where ISO- or ASCII-coded character sets are used. The choice between the alternative methods will depend on whether the data are presented in characters or larger units. For use in financial messages a standard is available[9] giving more detail.

For the generation of an authenticator in CBC mode, the message to be authenticated is enciphered as a chain of 64-bit blocks, any incomplete final message block being padded out with zeros. The generated ciphertext is not transmitted. The authenticator is derived from the final output block, taking the most significant m bits. No additional DES operation is required at the end of the process because it is not possible for an enemy to make a change in the plaintext and a compensating change in the authenticator. Figure 5.4 shows how the process of CBC authentication operates. All the cipher blocks except the last are used only for feedback. The most significant m bits of the last block C_n is used to provide the authenticator. In US standards the authenticator is sometimes called the *message authentication code* (MAC).

In CFB mode the text to be authenticated is subjected to a CFB encipherment process. When all the data have been enciphered and the last unit of ciphertext fed into the DES input register (completing the CFB operation as normally understood), the DES device is operated once more, producing a new state of its output register. From the output register, the most significant m bits are extracted to form the authenticator. Figure 5.5 shows this method; note that if the m bits of MAC were taken from the DES input register at the end of the text

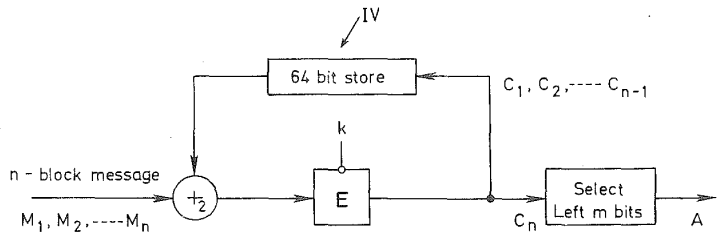


Fig. 5.4 Authentication by the cipher block chaining mode

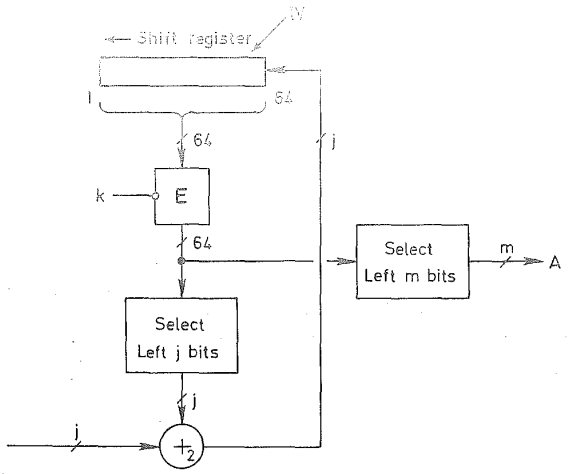


Fig. 5.5 Authentication by the augmented cipher feedback mode

encipherment, then for large enough m it would be possible to make compensating changes in the last plaintext character and the authenticator, hence the extra operation of the DES. This CFB method is not much used. The CBC mode is used in practice.

The number of bits used in the authenticator is subject to agreement between the communicating parties; it should be as long as is practicable. For US Federal telecommunications applications it is recommended[8] that the authenticator should be at least 24 bits in length. For financial purposes the US standard recommendation[9] is that m should be 32. m should be large enough that the risk of a matching value occurring by accident with a bogus message, which is 2^{-m}, is small enough. Making m longer than necessary would add to the message overhead due to the A value it carries.

The authenticator values obtained either by CBC or CFB mode depend on the entire message stream or chain. In any application the start and finish points of this stream or chain must be defined and also the location, size and format of the authenticator field. The authentication depends also on the IV which is loaded into the 64-bit register at the start of the operation. It is possible to use a secret IV, as is customary for CBC encipherment. However, the purpose of the IV is to prevent codebook attack on the first part of the ciphertext and this is not relevant to authentication. The simplest convention is to use a zero IV and this is adopted in several standards.

5.6. MESSAGE INTEGRITY BY ENCIPHERMENT

Encipherment of a message offers, in itself, a kind of integrity. If the sender and receiver of a message share the secret key, receipt of an enciphered message which deciphers into a sensible plaintext message gives the receiver some assurance that the message has arrived without unauthorized changes and came from the other owner of the key. 'Sensible' in this context implies that the message contains

sufficient redundancy. When an authenticator is used, this provides the redundancy. Nearly all communications carry redundancy, but it is not always easy for a receiver's processor to test for correctness, so in practice redundancy is added to messages when encipherment is used for authentication. The extra redundancy is a new field in the message sometimes called a 'manipulation detection code' (MDC). The choice of a method for calculating MDC values, which is not a trivial matter, is considered in the next section.

An attempt by an intruder to alter selected bits of ciphertext generated in CBC mode leads to random changes in the deciphered message, which should be obvious to the recipient. A single bit alteration in one ciphertext block will result in random changes in the corresponding plaintext block; note, however, that the exactly corresponding bit will be affected in the next plaintext block. A garbled plaintext block in a CBC decipherment should be taken as a warning to mistrust the plaintext of the next block. It is safer to discard the whole message and request complete retransmission. If the transmission is in CFB mode, then the intruder can also select the plaintext bit he wishes to alter, but his action should be apparent to the receiver because of the error propagation property of CFB encipherment; at least sixty-four following bits will be subject to unpredictable change. The final character of a CFB transmission can be altered without detection since there are no following characters to betray the change. It should never carry vital data. In CFB the plaintext unit preceding a garbled section should be mistrusted and retransmission is again the safest course.

Choice of the plaintext sum check method for authentication

Drafts of the Proposed Federal Standard 1026[10] which contained a section on encipherment at the data link layer proposed a sum check on plaintext called the 'manipulation detection code' (MDC). This check field would be produced by splitting the message or 'chain' into consecutive 16-bit words (the last one padded with zeros if necesary) and adding these words together modulo 2. The result was a 16-bit field that was appended to the plaintext before encipherment; an alteration to the ciphertext was expected to be very likely to upset the relationship between text and MDC on decipherment, thus detecting an active attack. The proposed FIPS for 'Data Integrity' made a similar proposal. Unfortunately this choice of sum check method was bad and it will be replaced by a better method. The reason for this weakness is instructive and will be described.

Consider first the CBC method, because this is the easiest to analyse. Figure 4.2 of chapter 4 shows how a ciphertext chain $C_1, C_2, \ldots C_n$ is deciphered. If the IV is I, the resultant plaintext chain is

$$I +_2 Dk(C_1),\ C_1 +_2 Dk(C_2),\ \ldots\ C_{n-1} +_2 Dk(C_n).$$

The modulo 2 sum of these 64-bit words is

$$I +_2 \sum_{i=1}^{n-1} C_i +_2 \sum_{i=1}^{n} Dk(C_i).$$

If a modification can be made to the ciphertext chain which leaves this 64-bit sumcheck unaltered, it will also leave the modulo 2 sumcheck of 16-bit words

unaltered. Such modifications are easily made. Any pair of blocks can be interchanged since this only alters the sequence of blocks in both sums, except that the last block C_n cannot be used because C_n does not occur in the first sum. Any ciphertext block can be inserted into the sequence twice, adjacent or not, since equal blocks cancel in both sums, if the end of the sequence is avoided. Changes of this kind do not alter the 16-bit modulo 2 sum and will not be detected.

Cipher feedback is a little more complex, but an example of a modification which escapes detection can be given. Assume that 8-bit CFB is used, so that eight characters fill the shift register. Then the text consisting of the character sequence

A B C D E F G H

can be modified to

A B C D E F G X A B C D E F G X A B C D E F G H

without changing the 16-bit sumcheck. The proof of this is left as an exercise for the reader. The method of proof we used for CBC will serve, but the eight characters of IV must be assumed to precede the given character streams and these will enter into the calculation.

It is a simple matter to avoid this problem by addition modulo 2^{16} for the 16-bit word, that is to say ordinary addition with suppression of carries beyond the sixteenth bit. It is true that a ciphertext can be introduced if it is inserted 2^{16} times, but this is neither likely to be useful nor to go undetected.

It has not usually been suggested that a modulo 2 sumcheck can be applied to the output feedback mode of encipherment, since compensating modifications are so easy to produce. Nor will modulo 2^{16} arithmetic avoid the problem entirely because changes to the most significant bits of the 16-bit words add by modulo 2 addition without affecting other bits and these changes can be used freely if an even number of them are changed. This problem can be overcome by using modulo $2^{16}-1$ arithmetic, in which carries beyond the sixteenth bit are cycled round to add into the least significant bit. To add with modulus $2^{16}-1$ the 16-bit words are added into a register large enough to avoid losing carries. Assuming that a 32-bit register is long enough, the final step is to add the top 16 bits to the bottom 16 bits. If a carry is produced this carry is added into the least-significant bit (and no further carry is produced).

This example of the faulty MDC illustrates that it can be difficult to find a system's vulnerabilities. The methods using modulo 2^{16} and $2^{16}-1$ arithmetic have some weaknesses, but not, we hope, serious ones..

Encipherment or authentication?

We have seen that encipherment with a secret key provides one form of message integrity; it is now fair to ask: Why should it be necessary to consider any other method? Why not encipher with a secret key all messages that require authentication? The reason is that, whereas authentication may be required in a given context, encipherment may be an embarrassment.

Consider a system in which mesages are broadcast to many destinations, one of which has the responsibility of monitoring authentication. Here transmission must be made in plaintext, with an associated authentication field. One terminal checks the authentication field and holds the key for this purpose. The other ter-

minals are simpler in design and possess no authentication capability. They rely upon a general alarm being given by the special designated terminal if an authentication failure is detected.

Another possible scenario involves a point to point communciation in which the receiving terminal has a heavy load of processing responsibilities, such that it cannot afford to decipher all received messages. Authentication is carried out on a selective basis, messages being chosen at random for checking.

Yet another very important application of authentication of a plaintext is that of a computer program text. Here it is essential that the text shall be capable of interpretation by a processor; decipherment each time the program was required to run would be wasteful of resource, therefore authentication by full encipherment is inappropriate. It is possible to authenticate the program text by appending an authentication field. This can be checked whenever assurance is required of the integrity of the program.

Authentication without a secret key

Neither encipherment nor authentication can be provided 'out of the blue'. The methods we have discussed so far have depended on a secret key, common to both sender and receiver. It is also possible to base authentication on an *authentic* number. This is no more paradoxical than basing the secrecy of enciphered data on the secrecy of a key because what has been achieved is that a large message is authenticated by a much smaller one.

In place of the authenticator A(k,M) let us use a 'one-way function' A(M) of the message alone. This must have the property that if a set of messages M is known together with their A(M) values it is nevertheless exceptionally difficult, given some other value A_0, to find *any* message M such that $A(M)=A_0$. A necessary condition is that the value A must have a large range, for example it could have 64 bits, large enough to prevent a table of messages being compiled and sorted on their A values so that for most A values a message could be found. Such a function is sometimes called a 'hash function' but these 'pseudo-random' functions are used for several purposes and some hash funcitons are not suitable for authentication. An example of a suitable function appears in chapter 9 (page 264).

To authenticate message M, the quantity A(M) is calculated and sent to the entity receiving or reading the message n such a way that A(M) is certainly authentic. With its aid M can be checked to see if A(M) agrees with the authentic value. If it does, M is pronounced authentic since it is assumed that no enemy could have found a bogus M that fitted with A(M). Applications for this kind of authentication are not widespread, but examples will appear later. Suppose, for example, that Ann has sent Bill a large file of data and Bill wants to ensure its authenticity. If both can agree on a one-way function A(M) then a telephone call will enable Bill's A(Mb) to be checked with Ann's A(Ma). If they are equal it is a good bet that Ma=Mb, so Bill can accept his version. The telephone call is assumed to confer authenticity because each knows the other's voice and can chat about mutual interests which would betray an impersonating enemy. Note that methods which do not use a secret key are vulnerable to the 'birthday attack' described in chapter 9, section 9.3.

Whether authentication is by encipherment or the use of an authenticator field, the defence is not complete because an authenticated or enciphered message can be read by an enemy and later used as part of a bogus transaction, provided this is done during the lifetime of the key. It is this possibility of 'message replay' which complicates authentication. Replay can take the form of changing the message sequence or replacing a stored data item by a value it held previously. In this section we shall consider the set of messages which make up a transaction between two communicating 'entities'.

The solution to the problem of replayed messages is quite simple in principle; it requires that each message should be different from all preceding messages using the same key and that the receiver should be able to test the 'newness' of each message. Furthermore, the sequence of messages which together form a transaction must be linked in some way so that their sequence can be checked. Protection against message replay would be complete if each communicating entity compared its received messages with all those it had previously received with the same key and rejected any messages which were copies, but this would be laborious and cannot be a general solution to the problem.

Authentication of transactions is basically a two-way requirement with each entity needing to authenticate the other. There are applications in which one-way authentication is sufficient, but the need for two-way authentication must always be considered. There is no longer any economic reason for transactions to do without two-way authentication.

In the conventional exchange of messages, each message is a response by an entity to the previous message which it received from the other. Typical short transactions employ one, two or three messages and there are circumstances in which a three-message protocol is essential for two-way authentication.

Use of a message sequence number

Two entities which frequently exchange transactions can maintain a sequence number n which increases by one for each message they exchange. Figure 5.6 shows how this number would be used in an exchange of messages M_1, M_2 ..., with authenticators in each message. Each message contains a sequence number and each entity keeps a note of the next number it will send and the next number it expects to receive. The authenticator is computed over the whole of the remainder of the message, including the sequence number, so neither the message content nor the sequence number can be falsified without detection. Since each party knows which sequence number to expect next, deletion of messages, replay of previously transmitted messages or changing the sequence of messages is detectable by the receiver.

The requirement to link together the messages which make up a transaction can be met in other ways. Figure 5.7 shows a scheme in which the authenticator of each message is repeated in the following message. Message M_2 contains a copy of the authenticator A_1. Its own authenticator A_2 covers both the message content and A_1. For practical purposes, the authenticator is a random number which links together or 'chains' the sequence of messages so that it is extremely

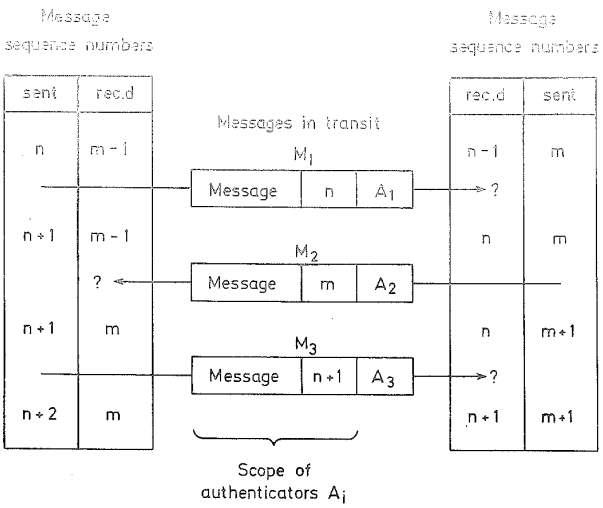


Fig. 5.6 Sequence numbering an exchange of messages

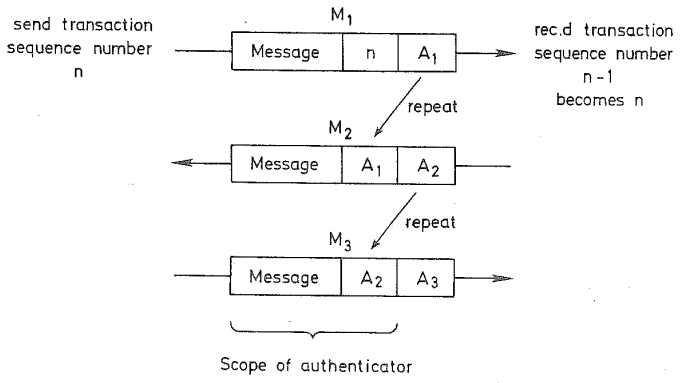


Fig. 5.7 Chaining messages by their authenticator values

unlikely that a false chain can be assembled by an enemy. The size for the linking authenticator is chosen with the criterion that the probability of an enemy finding a link by accident is too small to make it worthwhile to wait and search; typically from 16 to 32 bits are used.

Where messages are enciphered this provides a convenient way in which messages may be chained together by making the process of CBC run over from each message to its response. Thus the final 64 bits of each message become the initializing variable of the succeeding one. This form of chaining does not occupy space in the message. A similar method of chaining can be used with CFB.

When messages are chained in one of these ways (by repeating the authenticator or deriving the IV from the previous message) the sequence number n in the first message identifies the whole transaction. Supposing that transactions

can be initiated by either entity it must be decided whether each maintains its own transaction sequence number or a common sequence is used for transactions initiated by either. If there are two separate transaction sequences, the origin of the transaction must be included in the first message, otherwise the enemy might record a transaction by A and later play this back to A as though it was a transaction initiated by B.

The sequence number can be initialized, for example at zero, whenever new keys are distributed. The field of n must be large enough to accommodate all the transactions which can take place before the next key distribution. In many systems this is an argument for changing the key frequently, to keep the size of the n field reasonably small. If, due to a communication or processing error, the sequence numbers held by the two entities get out of step, the resynchronization procedure must not allow the undetected insertion of stored messages by an enemy. This requires that the new value of n should be greater than any previous value held by either of the entities. Resynchronization according to this rule is simplified if the requirement of strictly incrementing n is dropped and the receiver merely checks that n has increased with each transaction he receives. This relaxed rule allows the sender, optionally, to hold a single sequence number for use in all the transactions originated (or messages sent), but this does not remove the need for each entity to keep a separate sequence number corresponding to each source for checking all received transactions (or all received messages).

The use of random numbers for entity authentication

In a large community, authentication can be based on session keys derived from a key distribution centre (see chapter 6), so that each entity does not need keys for all those which might communicate with it. Alternatively, public key ciphers (see chapter 8) can be used to simplify the key-distribution problem. In these circumstances, maintaining separate sequence numbers for each possible communicating entity is not usually practicable. The need to make each message different from others using the same key can be met (with high probability) by including a random number. Consider, for example, the three-message transaction shown in Figure 5.8. The originator of the transaction includes a random number R, which is returned as part of the response. Each message has an authenticator

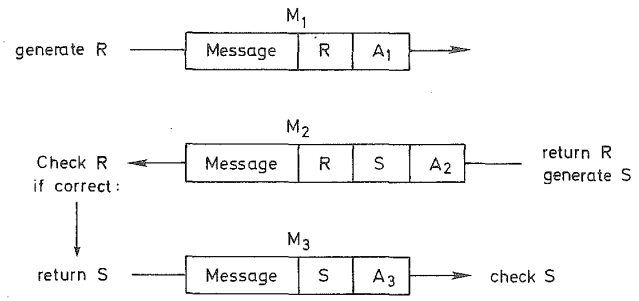


Fig. 5.8 Random numbers for entity authentication

covering all the content, including the random numbers, to enable tampering with the content in the communication network to be detected. The return of its random number assures the initiator that the response comes from a real entity and is not a recording of some earlier response. The size of the random number field is in this case governed by the Birthday paradox (see page 262). If there are likely to be N transactions using the same key, the random number field should be greater than N^2 in order to reduce to a low value the probability of a repeat.

For two-way authentication, the other entity includes in its response a random number S and this must be returned in the third message of the transaction as an assurance that the transaction is newly initiated and not a repeat. Each entity puts its own random number into the protocol in case the other is a masquerader.

Consider the possibility that M_1 is a replay of an earlier initial transaction message. The innocent receiver returns M_2, including the correct response to R and a new random number S. If M_3 is received in reply it must have come from an entity which possesses the secret key needed to generate the new authenticator value A_3. Could this be the genuine owner of the key if M_1 were a fake? This owner would not have accepted M_2 as valid unless it sent M_1. Therefore M_1 must have come from the genuine owner of the key and can be accepted as valid. But it is important that M_1 is not accepted as a valid message until M_3 is received and checked. For example, if M_1 were a request to update a data base, this action should be deferred until M_3 arrived.

If the transaction were to continue with further messages it would be necessary to chain them to the earlier ones by one of the methods we have described earlier.

Because of the Birthday paradox, the use of random numbers approximately doubles the size of field required for a given life of key. This presents no problem when a session key is used for just one transaction because the random numbers then have to be no bigger than normal authenticators.

The use of date and time stamps

Transactions employing a single message are vulnerable and should be avoided if possible. Where they are used, there is no possibility of the sender authenticating the receiver, but the receiver should be sure that the message came from the sender directly and was not a recording of an earlier message. A sequence number can achieve this but, in a large group, maintaining sequence numbers could present almost as much difficulty as storing all the received messages for comparison with new ones.

Where message transmission is reasonably rapid, an assurance of the 'timeliness' of the message is sufficient to prevent undetected message replay by an intruder. Store and forward systems delay messages, but do not prevent this method of checking if they can be relied upon to keep messages in sequence, since the time and date stamp provide a kind of sequence number which is universal for all messages received. Circuit-switched networks give delays less than 1 s and modern packet-switched systems should not exceed this figure very often; therefore, unless the rate of transactions is very high, a date and time stamp with a precision of 1 s is adequate.

If the date stamp is allowed to repeat, there is a risk of replayed messages. A very cautious view would be that no system is likely to be operating according

to the same procedure for 100 years, so a date and time stamp containing a two-digit year, month, day, hour, minute and second is certainly adequate. Each of these fields has two decimal digits and when expressed in hexadecimal characters this stamp occupies 48 bits. Employing a binary date would occupy a 16-bit field and a binary count of seconds in the day would use 17 bits. With these conventions a comprehensive date and time stamp would occupy 33 bits.

A simple quartz crystal clock provides a timing standard maintainable with 1 s accuracy, but, allowing for the need to correct it from time to time and the combined tolerances of two clocks, a margin of 5 s is likely to be the tightest criterion in general use. This can allow replays to be detected in all but the busiest systems. If public data networks provide a digital date and time source, this does not always mean that local clocks can be dispensed with, because of the risk that an enemy might be able to interfere with the time signal received. The public date and time source can safely be used to correct the local clock, with a warning if the correction is too large.

In theoretical treatments of the synchronization problem, much trouble has been taken to invent time criteria which do not require strict synchronization between the co-operating entities. With modern clocks and communication networks, 'real-time' can be re-instated.

In the very simplest terminals there may be cost criteria preventing the use of clock time, otherwise it presents no great problem. Active tokens, such as microcircuit cards, will usually have to accept a date and time value given by the terminals into which they are inserted and the security implications of this trust have to be examined.

Integrity of stored data

A file of data can be protected against illicit modification using authentication techniques closely parallel to those used for transmitted data. Authenticator values are calculated, using a secret key, and stored along with the parts of the file they protect. Users of the stored data also hold the key value and can check the authenticator.

The entire file could be protected by one authenticator, but any user would have to make the calculation over the whole file before using any of its data. Any change to the file, however small, would entail recalculating the authenticator over the whole file. It is better to divide the file into convenient 'units' which tend to be read or written at one time. Most files lend themselves to this kind of subdivision. But then a new threat appears — the units could be rearranged by an enemy if their authenticators are carried with them. If each unit has a built-in identifier, like the name and address in a file of data about people, this may not be a significant attack. If, however, a file is accessed by position (i.e. file address) the rearrangement threat is real. It can be met by including the address in the authenticated data. This does not require the address to appear in each stored unit, the address is inserted only for the purpose of calculating and checking the authenticator.

Messages and transactions are subject to the threat of a 'replay' during the currency of one key. This has a counterpart for stored data in the threat of 'restore'. After an item has been (legitimately) altered it can be restored by an enemy to

its old value for which the authenticator is known. The attack could be an active attack on the stored data or it could be applied to a communication channel by which these data are accessed. The key used for authentication of stored data probably has a relatively long life because changing it requires a replacement of the authenticator values and that means processing the entire file. To protect against this threat we need a sequence number or a date and time stamp. Unless the units are large, adding date and time is an excessive overhead, but it might be a natural part of their contents. The sequence number becomes a *version number* which each unit carries in addition to its authenticator. If units are updated piecemeal all their version numbers will be different and there would be no way that a reader of the file would know that the latest version was being read. When the date and time or sequence number are not natural parts of the file, yet piecemeal updates are needed, there must be a separate 'update record'. This is a record which has its own authenticator and includes a date and time stamp to verify it has not been 'restored' by an enemy. It contains, for each unit in the file, a note of the current version number. It is not necessary to store the version number also in the item itself, but the number must be part of the authentication function for that unit. Therefore checking the authenticity of one unit of the file requires this procedure.

1. Check the authenticator and date/time stamp of the update record.
2. Read the version number of the required unit.
3. Format the version number (and the address, if used) with the contents of the unit.
4. Check the authenticator of the unit.

The use of a separately authenticated update record illustrates an hierarchical approach to authentication which we shall meet later in the application of digital signatures — an advanced form of authentication which has particular relevance to stored data.

5.8. THE PROBLEM OF DISPUTES

Authentication using a secret key is symmetrical — both parties know the key and either can generate an authenticator value. The secrecy of the key protects them both against third parties but not against each other. Disputes can arise between them.

Ann may send Bill an authenticated message, but if Bill is dishonest he can forge a different message and claim it came from Ann. The authenticator value provides no evidence to resolve a dispute, for it could have been given by Bill to a false message. Since it is feasible for Bill to falsify a message, Ann can deny sending a message she really did send. Bill cannot prove it is genuine by producing the authenticator value. Business is possible between honest people who trust each other, but there is no way to adjudicate disputes, even if the judge is given the value of the key. This problem — the inability to provide evidence to resolve disputes — is called the 'problem of disputes'. To avoid this is the purpose of the OSI service of 'non-repudiation'.

The most elegant solution to the problem of disputes is the 'digital signature' described in chapter 9, which continues this account of authentication.

REFERENCES

1. *Data processing — check characters.* Draft International Standard 7064, International Organisation for Standardization.
2. 'Data Transmission Over the Telephone Network: Series V Recommendations', *The Orange Book,* VIII.1, International Telecommunications Union, Geneva, 1977.
3. Davies, M.C., Barley, I.W. and Smith, P.E. 'The effect of line errors on the maximum throughput of packet type computer communication and a solution to the problem', *Proc. 6th International Conference on Computer Communication,* London, 943, September 1982.
4. Nacamuli. A.I. 'Integrating the processing of foreign exchange dealings', *Proc. International Conference on Computers in the City,* London, 317, May 1983.
5. Linden, C. and Block, H. 'Sealing electronic money in Sweden', *Computers and Security,* 1, 3, 226, 1982.
6. *Banking — Requirements for Message authentication (wholesale),* International Standard ISO 8730, International Organization for Standardization, Geneva, 1987.
7. *Banking — Approved algorithms for message authentication Part 2: Message authenticator algorithm,* International Standard ISO 8731–2, International Organization for Standardization, Geneva, 1987.
8. National Bureau of Standards *DES Modes of Operation,* Federal Information Processing Standards Publication 81, December 1980.
9. *Financial Institution Message Authentication,* ANSI Standards Committee on Financial Services, ANSI Standard X9.9, August 1986.
10. National Communications System *Telecommunications: interoperability and security requirements for use of the Data Encryption Standard in the Physical and Data Link layers of data communications.* Federal Standard 1026, 3 August 1983.

Chapter 6 KEY MANAGEMENT

6.1. INTRODUCTION

The security of encipherment depends on protecting the key. In effect, encipherment concentrates the risk of discovery on the key in order that the data can be handled more easily. For this reason, the management of cryptographic keys is a vital factor in data security. Keys used for authentication and data integrity must be handled with equal care. In this chapter we describe some key management methods designed for the two main applications of encipherment, storage of files and communication of messages. In order to cover the subject in reasonable depth we have taken as examples two well-developed schemes, one coming from International Business Machines (IBM)[1] and the other an international standard.[2]

In a large network, messages are transferred between many terminals and hosts, on behalf of many users. The essence of end-to-end encipherment is to keep these users and these terminals separate so that they cannot read or interfere with each other's messages except where they are allowed to, by the sharing of keys. There will be many encipherment keys and their management is complex. Similar problems are presented by a host computer which stores files for different users and is asked to communicate some of these, under strict control to other users. A key management scheme should be able to handle a multi-user, multi-host and multi-terminal situation and protect data both in storage and in the communication network.

Key management includes every aspect of the handling of keys from their generation to their eventual destruction. The main complexities occur in the distribution of keys and their storage, so these will occupy most of our attention.

In order to move a key through a communication network, it must be enciphered with another key. For example, if we have a key *ks* which is used to encipher data and need to transport this key through the network, we use another key *kt* to encipher it as E*kt*(*ks*). There are advantages in using a data enciphering key like *ks* for only a short period and then establishing a new one through the network.

Since the key *ks* is typically used to encipher data for just one session it is called a *session key*. When the session key is being moved out to a terminal, the key *kt* used to encipher it for transit is called a *terminal key*. A terminal key is used for a longer period than a session key and it may have to be stored at a host computer with a number of similar keys for different terminals. In order to minimize the amount of secure storage needed, all these can be enciphered under yet another key *km* which is called a *master key*. Thus the storage of *kt* is in the form E*km*(*kt*).

The master key protects the secrets of the terminal keys and these protect the data-enciphering keys *ks*, which in turn protect the data, in a kind of hierarchy with the master key at the top. In their protected form, the lower keys (and the enciphered data) can be stored in ordinary memory or transmitted through a communication network. The whole system then depends on the master key which therefore needs the greatest attention to its security. Usually it is stored in a physically secure box. The key hierarchies of the IBM and ISO 8732 schemes will be described later.

6.2. KEY GENERATION

An ideal method of key generation would be one that chose the key at random — with an equal probability of choosing any of the possible key values. Unfortunately, this is difficult to achieve and we often have to make do with less. Any non-randomness which could give an enemy some way of predicting a key value or finding a value with higher than normal probability would reduce the task of searching keys and make cryptanalysis easier. Fortunately, a completely uniform distribution of key values is not essential; unpredictability is the essential requirement.

A classic method of generating random numbers is tossing a coin or throwing dice. It is perhaps surprising to find that methods like this, which are labour intensive, are suggested by IBM[3] for generating the top-level master keys. In fact, the labour involved is not significant because these keys are very rarely changed. The reason for choosing manual methods is to have them completely under the control of a trusted individual. Any mechanism or algorithm that is provided for generating keys carries with it a danger that someone has arranged the mechanism or algorithm to generate predictable, though apparently random, numbers. Tossing coins or throwing dice has the advantage that the operator has the whole procedure under his control. A small bias that might be introduced by wear on the coin or die would not significantly change the security. The IBM authors[1] publish a table for convenient conversion from the results of tossing a coin seven times to the two hexadecimal characters which represent one octet of a DES key. Fifty-six throws are needed for the whole key.

The top-level master key has such importance that the methods of double or treble encipherment described in chapter 3 are sometimes used. By having different people generate the various keys, the responsibility for safe generation of the master key can be spread.

Manual key generation is not suitable for the large number of keys in the levels below the master key. Some procedures need new keys very frequently. There are two possible sources of the random numbers required for numerous keys, a genuine physical source of random bits or pseudo-random number generators.

Random bit generators

Electrical noise is a problem in amplifier design but can be turned to good use. All resistors generate noise (though less at very low temperatures) and a wide-band amplifier will turn this noise into a signal which can switch a gate on and off. One method of generating random bits uses the zero crossings of this signal

to produce pulses which change the value of a single-digit binary counter. The intervals between these pulses will vary in a random way and their probability distribution depends on the pass-band of the amplifier. At uniform time intervals the state of the binary counter is sampled to give the value 0 or 1. If the method works according to plan, this should be a random binary digit with its value uncorrelated with any other digit that has been produced. Careful design is needed to minimize bias or correlation. In order to remove any residual flaws of this kind, the random data can be used as key or data (or both) in a cipher algorithm to obtain unbiassed, uncorrelated results. This is the best source of random data for keys and the equipment cost is small.

Pseudo-random number generators

There are many examples of mathematically determined sequences which generate digits that are, to all appearances, random. Unfortunately, when a variety of statistical tests are tried, many of these 'pseudo-random number generators' can be shown to have a pattern in their output. Fortunately, a good cipher algorithm helps greatly in generating pseudo-random sequences. If the algorithm works well as a cipher then almost any method of using it to generate a sequence should generate random numbers, otherwise there is a pattern in the cipher output which would be a weakness in the cipher.

For example, if the key, k, is chosen then the result of enciphering the sequence of natural numbers 0, 1, 2, ... should generate a sequence of random numbers, each one unrelated to its predecessor. In other words, a random number sequence R_n can be generated as

$$R_n = \mathrm{E}k(n).$$

There is a danger that two sequences generated in this way might use the same key and therefore be identical. This illustrates the rule that random number generators should have random data to start them off. They use numbers called 'seeds' because a series of random numbers can be grown from them. A series grown from seed values cannot be infinite in length but it can be very long. The example we have just given does not stop generating random numbers until the value $n=2^{64}$ is reached.

The purpose of our random sequence is to generate key values (and perhaps initializing variable (IV) values) for a key management scheme. The essential requirement is that the keys it produces should be unpredictable, even if many of the earlier keys are known. The simple random number generator we described might not be able to conceal the value of n which entered into its calculation but, if the security of the DES cipher is good enough, this still does not allow the seed value k to be deduced and so none of the other random numbers can be found. It would not be wise to keep a key generator of this kind operating indefinitely with the same seed value. There must, therefore, be a source of seed values, obtained by a more reliable random process, such as tossing coins, throwing dice or the random noise source.

Most systems use random number generators that are a little more complex than our example, using many different sources of randomness or unpredictability, for extra safety. One source of unpredictability is a secret number which is already

in the system such as a top-level master key. Though this master key is well protected against discovery, it is used in the cryptographic operations of the system. A pseudo-random number can employ these operations, which are not available to an enemy outside the computer system.

An an example, the seed values used in generating the random sequence can be obtained by cryptographic operations (using the master key) on a number representing the current date and time.

A useful source of randomness is the statistical nature of the traffic handled by a host computer. For example, the number of calls on the operating system can be counted. It would be difficult to predict this number exactly from outside the system. An unpredictable quantity can be obtained by timing an operator's reaction. When the system is started, it can ask the operator to pause a while and then press some convenient button. If the elapsed time before pressing this button is measured in microseconds, the range of possible values certainly extends over several thousand. By putting together a calculation employing as seeds a number of unpredictable or random quantities like these, it is possible to generate a number sequence which is for most practical purposes random.

An example of a pseudo-random number generator designed according to these principles is given in reference 3. The notation has been changed to simplify the appearance of the expressions. It has a starting value U_0 and generates a sequence U_i, making use of cryptographic functions which employ two secret master keys, shown here as k_1 and k_2. The generation of the sequence employs a further sequence of seed values Z_i each derived from some real-time clock readings with an input-output operation of unpredictable length between successive clock readings. The calculation of the sequence U_i employs these relationships

$$X_i = \mathrm{D}k_1(U_i)$$
$$U_{i+1} = \mathrm{E}k_2[\mathrm{D}X_i(Z_i)].$$

The random number sequence R_i is obtained from the U_i by the relation

$$R_i = \mathrm{E}k_2[\mathrm{D}X_i(U_i)]$$

The complexity of this method makes it extremely unlikely that anyone could predict its outcome or deduce anything useful from a sequence of R_i values that it creates. It is perhaps a matter of judgement how complex a random number generator is needed in practice.

Figure 6.1 shows, in block diagram form, the operation of the random-number generator we have described. The register is shown containing the value of U_i before returning it for the next iteration. The two compound operations shown in the broken rectangles are the cryptographic functions employing the secret master keys shown here as k_1 and k_2. This is the operation which we shall meet later in the IBM key management scheme and which is known as 're-encipher to master key'. It is provided for a specific purpose in the key management scheme and has been adopted in the random number generator simply because it employs the master keys and is available to the operating system but not to the users of the host computer.

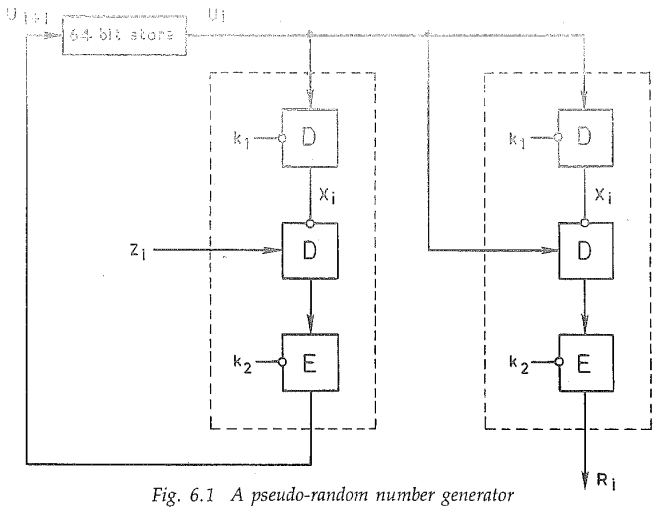


Fig. 6.1 A pseudo-random number generator

6.3. TERMINAL AND SESSION KEYS

Imagine a set of N terminals which we shall call T_i and that each terminal may wish to communicate in secret with any other. If all the keys to provide for the possible connections were produced in advance, each terminal would need to store $N-1$ keys and, supposing that the same key could be used for both directions of communication between a pair of terminals, there would be $\frac{1}{2}N(N-1)$ keys in total.

This method is clumsy and becomes more so as the population of terminals increases. In practice, few terminals are actually communicating at any time. The period of communication between two terminals is known as a *session*. It is better to provide keys only for those sessions which are actually in operation and these are the *session keys*. Then a procedure is needed to establish session keys whenever they are required.

The use of a session key has a worthwhile bonus of security. Because it exists only for the duration of one session, an enemy who could obtain one of these keys gets limited value from it. Of course, the establishment of session keys requires a key at a higher level in the hierarchy, in this case the terminal keys kt_i are employed. These keys are, in fact, better protected than the session keys because session keys can be attacked by a knowledge of the data that are being communicated, but terminal keys are used only to encipher session keys, so the information they encipher is both less in quantity and more random in nature than the data handled by the session keys. Effectively, an attack on the terminal keys would require a knowledge of the session key values so cryptanalysis must be successfully carried out twice to reach a terminal key.

When a session key, ks_{ij}, is needed for the communication between terminals T_i and T_j it can be transmitted through the network under two different encipherments, for these two terminals, $Ekt_i(ks_{ij})$ and $Ekt_j(ks_{ij})$ respectively.

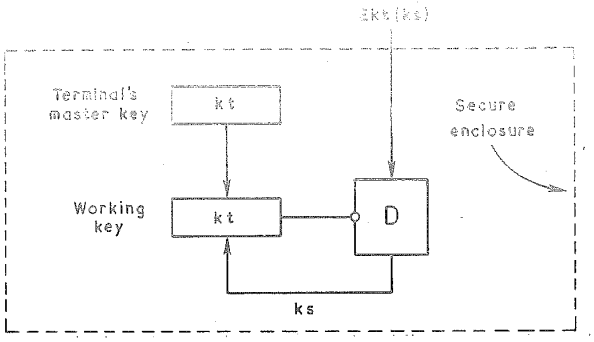


Fig. 6.2 Loading a session key at the terminal

These quantities are generated at some central point, which might be a host computer responsible for the operation of these terminals, or in a wider network there might be a specially appointed 'key distribution centre',[4] sometimes called a network security centre when it takes on additional responsibilities.

Figure 6.2 shows how the arriving session keys are handled at the terminal. The decipherment operation shown would typically employ a DES hardware device or chip which was capable of storing two keys. One of them, *kt*, is regarded in this context as the terminal's master key. It is held in a register where it can be made available each time a session key is to be loaded. Then its value is copied into the working-key register. This register supplies the key for the DES operation used to decipher the incoming session key and produce the clear value *ks*. In a well-designed chip, the value *ks* should not leave the device but be transferred after the decipherment into the working-key register, where it displaces the value *kt*. In this way, the physical security of the chip protects the session keys, since they do not come out of the chip at any time.

There are two other key management operations at the terminal, the loading of the terminal key and the destruction of the working key. Loading the terminal key is part of the process of transporting this key to the terminal and is discussed later. Destroying the session key could either be treated as a separate operation or regarded as a special case of loading a session key in which a dummy value such as 0 is loaded. It is useful for the destruction of the session key to be under the control of the host or key distribution centre but, on the other hand, any command from the centre can be stopped by an enemy, therefore, as a fallback, the session key should be cleared locally at the end of the session if the centre fails to do so.

Routes for distribution of session keys

Figure 6.3 shows a pair of terminals for which a session key *ks* is to be established. The key distribution centre (which might be a controlling host) must share a knowledge of the key kt_1 in common with terminal 1, and the key kt_2 in common with terminal 2. The most direct routes for the session keys are those shown on the left of the figure. These may be inconvenient in practice. One terminal takes the initiative in setting up the communication path, in this case we suppose it to be terminal 1. Terminal 1 must first communicate with the

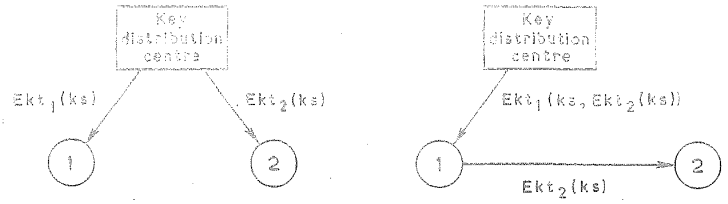


Fig. 6.3 Routes for session key distribution

distribution centre and request a session key, quoting the terminal 2 as the target for its future call. If the session keys are now distributed by the direct routes, terminal 2 will receive the session key before it receives the call from terminal 1. In fact it may be engaged in interaction with another terminal or unwilling to accept the call. At the time that it receives the enciphered session key it has no knowledge of which terminal is about to call it. In a busy situation, there will be a need for each terminal to be ready to receive several future session keys, storing them until a call arrives, then it must associate the call with one of these keys. Rules will be needed to decide the proper action in a number of error situations, such as a breakdown of terminal 1 before the call is completed, or a busy condition in the network which prevents the call. Time-outs could deal with these problems but time-outs do not produce neat and reliable solutions.

For these reasons it is usually preferred to transfer the session key to the target terminal (terminal 2) by way of terminal 1, as shown on the right of the figure. Terminal 1 must receive not only the enciphered form of *ks* for its own purpose but also the value of *ks* enciphered with kt_2, which it will pass on to terminal 2 when it is establishing the call. From the viewpoint of terminal 1, setting up this call has two phases, obtaining the key from the key distribution centre and then, after making the call to terminal 2, transferring an enciphered form of the session key to that terminal. In the figure we have shown that the key which is intended for transfer, namely $Ekt_2(ks)$, is included with the value of *ks* in the message sent to terminal 1 enciphered under kt_1. To do this, kt_1 should be used to encipher these data as a *chain*, to prevent an enemy separating the two parts for his own use.

This apparently simple key distribution function has been the subject of several security studies[5,6] which have considered a number of possible attacks and elaborated the protocol to frustrate them. There is a third possibility for routing the session keys, which is to send both keys from the distribution centre to terminal 2 and have terminal 2 relay the key required by terminal 1. Since the procedure begins with terminal 1 sending a request to the key distribution centre, this produces a complete cycle in which messages are passed successively from terminal 1 to the key distribution centre to terminal 2 and back to terminal 1. This circular route enables a random number chosen by terminal 1 at the time of making the request to be returned with the key he receives. By this means, a replay of old keys is prevented and the timeliness of the session key in use is ensured. This message sequence has not yet been elaborated and the examples we shall give below are of the kind shown on the right-hand side of Figure 6.3.

Session key distribution protocol

The two phases of key distribution are the interaction between terminal 1 and the distribution centre, called 'key acquisition' and the interaction between

terminals 1 and 2 at the beginning of the call which we denote as the 'key transfer'. In both of these operations, the encipherment prevents the critical quantity ks being discovered by an enemy, but it does not complete the protection against an active attack. We have to guard against 'spoofing' which is one terminal pretending to be another. For example, an enemy might manipulate the communication links to take the place of the distribution centre and provide terminal 1 with a key that was distributed earlier. This could be done without knowing any of the keys, merely by repeating earlier messages. Another threat is that an enemy terminal might impersonate terminal 1 to the distribution centre and obtain a new key for calling terminal 2. A third threat, which is more serious, is that an enemy terminal might masquerade as terminal 1 in calling terminal 2, using a key that had been transferred earlier.

The principal danger lies in an enemy re-establishing a session key that has been used before, either by masquerading as the key distribution centre and persuading terminal 1 to establish this key, or by masquerading as terminal 1 and establishing the old key for a fraudulent call to terminal 2. The same messages would be used that passed between the terminals when the key was used earlier, so there is no need for the enemy to know kt_1 or kt_2.

The trick of establishing an old key can be exploited in one of two ways. Suppose that the enemy had obtained the value of ks, then at any time that he could establish this value again, he could either listen in on a call between terminals 1 and 2 or impersonate terminal 1 in a call to terminal 2. Finding the value of ks might be the result of cryptanalysis applied to an earlier session and the ability to re-establish ks defeats one of the virtues of the session key which is that it is used only for a short time. Another way of exploiting the masquerading trick, even when ks is not known is to use it to replay an earlier interaction between terminals 1 and 2. This might enable a transaction between these terminals to be repeated with an advantage to the enemy. Of course, transactions have their own protocol which usually employs checks to prevent transactions being replayed. Nevertheless, it is legitimate to expect some help from the key distribution protocol in preventing such abuses of the system.

Authentication at the key acquisition phase

When it calls the key distribution centre, terminal 1 should establish that it is conversing with an intelligent device that knows its key value is kt_1, and is not simply an enemy repeating the data from an earlier transaction. This *authentication* of the key distribution centre can be carried out by giving the centre a task depending on kt_1. The following interaction will do this:

Terminal 1 sends to key distribution centre in clear

$$a_1,\ r_1,\ a_2.$$

Key distribution centre to Terminal 1

$$\mathrm{E}kt_1\{r_1,\ a_2,\ ks,\ \mathrm{E}kt_2(ks,\ a_1)\}.$$

The commas denote that the terms are parts of a continuous message. Where the message is enciphered it is treated as a chain but the IV can safely be zero, because each chain begins with a random number: a_1 and a_2 are the addresses

of the two terminals. They are sent with the request to the key distribution centre to identify the two terminals needing the session key. The quantity r_1 is a random number chosen for this interaction by terminal 1 and this same number appears in the enciphered reply, reassuring terminal 1 that it cannot be a recorded message, because it includes his random number. The other feature in the reply is the inclusion in the chain enciphered by kt_2 of the address a_1 of the calling terminal. When this reaches terminal 2, it helps to authenticate terminal 1 as the source of the message, because its encipherment with kt_2 shows that it must have come from the key distribution centre.

Does the key distribution centre need to authenticate terminal 1? Another terminal could make the call and present the same data a_1, r_1 and a_2. In fact, it would serve no useful purpose because in order to use the data from the centre it would have to decipher them with kt_1 which is a key known only to the key distribution centre and terminal 1. An intruder could change the value a_2 in the request and thus divert the subsequent transaction but the repetition of a_2 in the enciphered reply allows this to be detected. So authentication in this 'key acquisition' phase need only operate in one direction, in principle. But one factor might make it desirable for the key distribution centre to authenticate the calling terminals — the possibility that enemy terminals might overload it with bogus calls. The distribution centre can authenticate the calling terminal only by variable data, since fixed data can be replayed by masquerading terminals. The calling message could be replaced by

$$a_1,\ \mathrm{E}kt_1(n_1,\ r_1,\ a_2)$$

in this expression n_1 is a serial number for all the calls from terminal 1 to the key distribution centre since the key kt_1 was established. Its presence in the enciphered chain of this expression ensures that each call will be different and that the distribution centre can check the serial number, after decipherment, to verify that the caller possesses the key, kt_1. Alternatively a date and time can be used in place of n_1. The address a_1 must lie outside the encipherment so that the centre can identify the caller and use the correct key for decipherment.

There are other methods for authenticating the terminal to the key distribution centre but we question the need for further elaboration to solve this problem, since the only risk is the extra load on the key distribution centre.

Authentication at the key transfer phase

There is a clear need for two-way authentication between the terminals 1 and 2 enabling each to be sure of the other's identity and certain that it is not receiving a replay of earlier messages.

If terminal 1 is certain, because it authenticated the key distribution centre, that it received a new session key, ks, then the fact that the subsequent enciphered conversation with terminal 2 works will verify that terminal 2 is responding. No other terminal could respond sensibly because the establishment of the common session key ks requires it to decipher the key transfer message with its own key kt_2. If terminal 1 wants to get this reassurance early it can send another random number r_2 in the form $\mathrm{E}ks(r_2)$ and receive some function of r_2 in an enciphered message from terminal 2.

The authentication by terminal 2 of the calling terminal is more difficult. In order to detect a replay of a previous key transfer it can set terminal 1 a problem in the form of a random number r_3 transmitted as $Eks(r_3)$ and obtain a suitable function of r_3 in an enciphered reply. This does not guard against the masquerading calling terminal which has obtained the value of *ks* by cryptanalysis of an earlier exchange. Knowing the value of *ks* enables the bogus terminal to continue the masquerade.

Worse still is the case of the enemy discovering kt_1 because then he can obtain from the key distribution centre a stock of *ks* values for future masquerading. Even if terminal 1 changes its terminal key, the masquerade with old values of *ks* can continue, as long as terminal 2 keeps its master key unchanged.

The solution to this problem proposed by Denning and Sacco[7] is to time-stamp the session key. They propose the following squence.

Request from terminal 1 to key distribution centre

$$a_1,\ a_2.$$

Reply from key distribution centre to terminal 1

$$Ekt_1(ks,\ a_2,\ d/t,\ Ekt_2(ks,\ a_1,\ d/t)).$$

Key transfer from terminal 1 to terminal 2

$$Ekt_2(ks,\ a_1,\ d/t).$$

In these expressions, *d/t* is the date and time.

The value of *d/t* supplied to terminal 1 authenticates the key distribution centre. Any terminal which makes calls to the centre at shorter intervals than 1 min must keep a list of *ks* values for the last minute and check that these are not repeated. There will rarely be a need for such checking.

The value of *d/t* supplied with the key transfer to terminal 2 verifies the freshness of the session key. Keys obtained from the distribution centre must be used at once and cannot be stored for future use and the time required to establish calls must be at most a few seconds if the scheme is to work well. This immediacy should be easy with modern networks. Either terminal will reject a key that is too old, thus preventing replays.

Distribution of terminal keys

For the present, the possibility of non-secret methods of key distribution will be ignored (these are described in chapter 8) and we will describe more traditional methods using ciphers like the DES. Whatever key hierarchy is used, there must be at least one key at each terminal that has not been sent through the network. In the system we have just described this is the terminal key *kt*. Such keys must be loaded into the terminals manually and they may have to be changed from time to time for extra security. They must be physically transported to each terminal.

The older method of loading a key into a terminal was by rotary switches or a keyboard. This operation must be carried out only by trusted personnel, therefore a lock was usually provided into which a physical key was inserted and turned to allow the entry of a new key. Such key loading operations might be spied on by an enemy. If the key is written down on paper for its journey to

the terminal, it is difficult to be sure that the key has not been copied en route. For these reasons it is now becoming customary to load the key from a transport module in which it has been electronically stored.

The key carrying module, sometimes called a 'key gun', can be of pocket-calculator size and its essential external feature is the means of coupling it to the terminal to deliver its key and to the source of keys when it is loaded. The means of coupling can be electrical, but optical coupling is popular since it seems more difficult to tap an optical coupling.

A key transport module can carry a number of keys for each destination and can be used to load keys at several places on the courier's route. The keys are selected by an address given to the module. All that is needed in such a module is a suitable store such as an 'electrically alterable read-only memory' or EAROM. But this provides no security against the reading of keys while the module is en route. For a more secure system the key-transport module is constructed with a microprocessor and a battery so that a random access memory (RAM) can be used for storage, which is cleared under the control of the microprocessor when security requires it.

Suppose that each stored key has one destination and can be cleared by the module as soon as it has been read. (Where two stations need the same key this value is stored separately for each destination.) Then an attempt to read a key illegally elsewhere than at its destination prevents that key being used and achieves nothing. It is essential to prevent the key being replaced by the enemy after it has been read. This can be done by giving each key transport module a secret which it shares with the key generator. It could be an encipherment key but, more simply, the loading of keys into the module can be controlled by a password specific to the module. However, 'one-shot' key transporters are not always convenient. Some users prefer to keep their transport modules at the destinations to act as back up stores for the keys, in case the equipment receiving the keys fails.

When reading a key does not clear it, password control can be used to prevent illegal reading. Each destination prepares its own module by giving it a password. When the module has been taken to the key generator and received its keys, it will not give them up until it receives its password. If it receives the wrong password too many times it destroys its keys, to prevent a password search.

Two kinds of attack remain. One is opening up of a module to discover all that it contains and then successfully closing it to make it acceptable for use (or replacement by an acceptable replica). The other is the tapping of the communication path when it delivers the key at the terminal. Those present at the time of loading the key into the terminal must be vigilant enough to detect these attacks.

Given that the module resists physical attack or reveals it when it arrives at the destination, the module could possibly be transported by a postal or other carrier service to save the cost of special couriers.

6.4. THE IBM KEY MANAGEMENT SCHEME

In the remaining part of this chapter we shall describe two different key management schemes, both of which have been worked out in some detail. They are not only valuable as examples of comprehensive schemes, they also demonstrate

a range of techniques which can be used to develop key management methods for different applications. Like most working systems they contain a lot of detail and we shall not be able to cover every aspect.

The IBM key management system was described first in a group of papers in the *IBM Systems Journal*.[1,3] We have not used the notation of the IBM papers because we wanted to have a consistent notation throughout this book. In a few respects we have simplified the scheme to make it easier to explain without changing its essential features.

It is important to understand the environment for which the key management method was devised. Though it is like the key distribution scheme we described earlier, employing session keys and terminal keys, the environment is different because the terminals do not communicate directly with one another through a communication sub-network. Communication takes place through host computers, each host being responsible for a group of terminals. The host acts as a key distribution centre for its own group of terminals and encipherment is used for messages passing between the terminal and the host. There may be several hosts in a network and then the hosts organize the communication of messages or files between them. A key management system has two functions which the IBM scheme mostly keeps separate — the encipherment of messages to provide communication security and the encipherment of files to provide file security. Each of these functions has its own key hierarchy and the two key hierarchies have close parallels. We shall describe first the communication function and then the file security function, making comparisons between the two. The functions come together when files are transported from one host to another.

Physical security requirements

The handling of keys at the terminal is like that described in Figure 6.2. Each terminal contains a terminal key *kt* and at the start of a communication session, a session key *ks* is delivered to it, enciphered under the terminal key.

The physical security requirement at the terminal is the protection of the two keys. The loading of *kt* is a privileged operation. The key-management scheme must provide the value of *kt* for this purpose, whether or not a key carrying module is employed.

At the centre, the physical security requirements are much more severe. A number of terminal keys must be stored safely and at any one time many sessions may be in progress, so a number of session keys must be kept ready for use. The host computer is provided with a special, physically secure module containing all that secret information which must be protected. In the IBM papers it is called the 'Cryptographic Facility' and it is typified by the IBM 3848 cryptographic unit: we shall call it a 'Tamper Resistant Module' (TRM).

Both in the terminal and at the centre, the encipherment operation must be carried out in an appropriately secure place. At the host all encipherment operations occur inside the TRM. The TRM also contains the most important keys which, once they have been loaded into it, can never be read again. In the same way, none of the keys can be read from the hardware of the terminals.

The processing work of the host is carried out on behalf of many users, each with his own rights of access to certain terminals or files. As in all multi-user

systems there is an elaborate access control mechanism administered by the operating system which decides the rights of any *subject* with respect to any *object*. Subjects may be users or processes and objects may be data, files, programs, terminals etc. The right to use a terminal with encipherment means giving a subject the right to employ a certain session key.

The prevention of illegal access by one subject to another's objects is the responsibility of the access control system. Encipherment cannot improve on the security of this access control, it merely extends the security to stored files and distant terminals. An enemy within the system who can undermine the access control mechanism can steal files or read messages illegally in spite of the use of encipherment. Therefore, the key management system must rely on the access control method to control access to keys.

There is always the possibility that one of the system's users could be an enemy and could steal data from the system and pass it to an enemy outside. If he could steal keys and pass these to an enemy outside, then various stolen storage media, or ciphertext recorded from a line, could be deciphered and all the security of encipherment with the stolen keys would be lost. It is, therefore, a cardinal principle that the key management scheme should prevent anyone, including legitimate users and system programmers, from obtaining any cryptographic key. Therefore, cryptographic keys (with a few exceptions) are allowed outside the TRM only when they have been enciphered by other keys. As a result of this principle, the damage that can be done by an enemy within the system is confined to the data he reads by misusing the access control system of the host. He cannot steal keys that can be used outside the system. To put it another way, the barriers inside the system between the various subjects are not strengthened by the use of encipherment. The encipherment of data, together with key management, places a firm barrier between what can be done inside the system and what can be done outside.

The main principle is that no keys can be taken outside the TRM in their clear form, the form in which they could be exploited by an enemy. The exceptions to this rule are the terminal keys which must be in clear outside the TRM, at least at the moment when they are loaded into the terminal. This operation of loading terminal keys is carried out under careful discipline by trusted people. The procedure for initializing the whole system by the creation of top-level master keys and loading them into the TRM is another exception to the general rule and it requires the greatest discipline and trust.

The key hierarchy

There is a three-level key hierarchy which will now be described as it applies to the communications security part of the scheme. This description is intended only to give a general picture and the details will follow later.

The three levels are shown in Figure 6.4. At the top level are two master keys, $km0$ and $km1$. In the middle level are the terminal keys, denoted kt, one for each terminal belonging to the host. The lowest level keys are the session keys, ks, which are associated with terminals on a more temporary basis. Only the session keys encipher data. The terminal keys, as we saw earlier, are used to encipher session keys for transmission to the terminals.

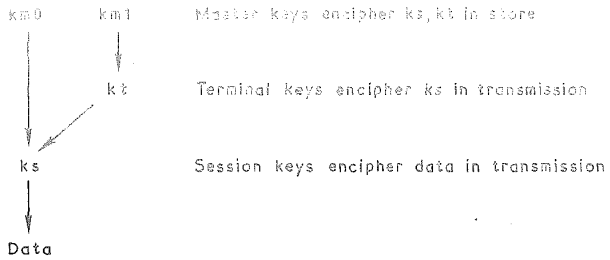


Fig. 6.4 The hierarchy of keys for communication security

The two master keys in the host, called *km*0 and *km*1, are used, respectively, to store all the session keys and all the terminal keys in the system. Because all the session keys are enciphered under the key *km*0 whenever they are outside the TRM, they can be stored in ordinary memory locations. The stealing from the host of a quantity such as E*km*0(*ks*) would not be disastrous because the master key *km*0 is special to this one host so the enciphered value of *ks* is useless outside it. Likewise, the terminal keys are stored as E*km*1(*kt*) in ordinary memory locations.

The purpose of storing the terminal and session keys outside the TRM is two-fold. First, it reduces the amount of store in the TRM and, secondly, it allows the control of access to these keys to be the normal access control provided by the operating system. There would be little point in having a duplicate or alternative access control system for keys.

The existence of the master keys inside the TRM makes all the operations of the TRM special to this one host. The various functions of the TRM are not all equally available. Some, such as the loading of the master keys, take place rarely and only with specially privileged people present. The operations of the TRM used for distributing new terminal keys are accessible only to the operating system. But at the lowest level, the operations of enciphering and deciphering data are available to ordinary user's programs, provided that they can gain access to the appropriate session key.

The encipherment and decipherment of data at the host

A program at the host computer can use the functions of the TRM to decipher data coming from the terminal or encipher data going to that terminal, if it can provide the TRM with the session key in use at that terminal. The operations carried out in the TRM are illustrated in Figure 6.5. They are not simple encipherment or decipherment because the key arrives in an enciphered form. The first action is to decipher the key using the master key *km*0 which is stored in the TRM. The resulting session key, *ks*, is then used for the encipherment or decipherment operation. This function of the TRM is given the name 'Host Encipherment' (HE) or 'Host Decipherment' (HD). In a sense, the host encipherment or host decipherment of each individual host is different because of the different master key contained in its TRM, so a session key taken from the store of one host cannot be used in another host.

The two inputs to the HE or HD operations must be distinguished because one is a key input and the other a data input. We have shown in our figure a small

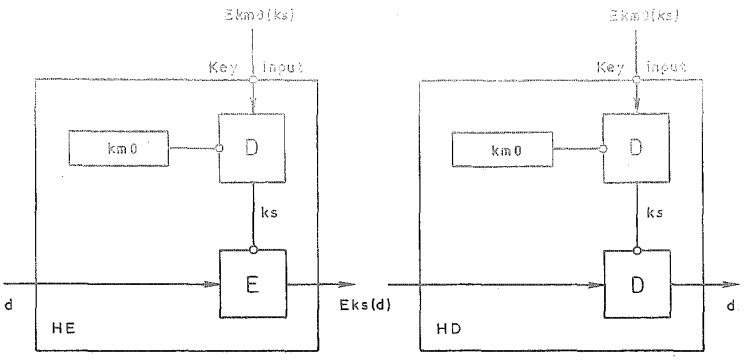


Fig. 6.5 Data encipherment and decipherment operations of the tamper-resistant module

circle where the key input enters the TRM by analogy with the notation used for the block cipher itself.

Generation and distribution of a session key

Figure 6.6 illustrates the function of the TRM which is used for distributing a session key. The new session key *ks* must be transmitted to the terminal in the form E*kt*(*ks*). Both *kt* and *ks* must be protected by encipherment whenever they are outside the TRM. The terminal key *kt* is stored under encipherment by the master key *km*1 and the first operation in this key distribution function is to decipher it and produce the plaintext value, *kt*. Then the stored session key E*km*0(*ks*) can be deciphered inside the TRM by *km*0 and re-enciphered by *kt* to produce the required quantity E*kt*(*ks*). This operation is known as 'Re-encipher from master key' (RFM) since it changes the session key from encipherment by the master key to encipherment by the terminal key. This function of key distribution is carried out by the host's operating system, responding to a request from the

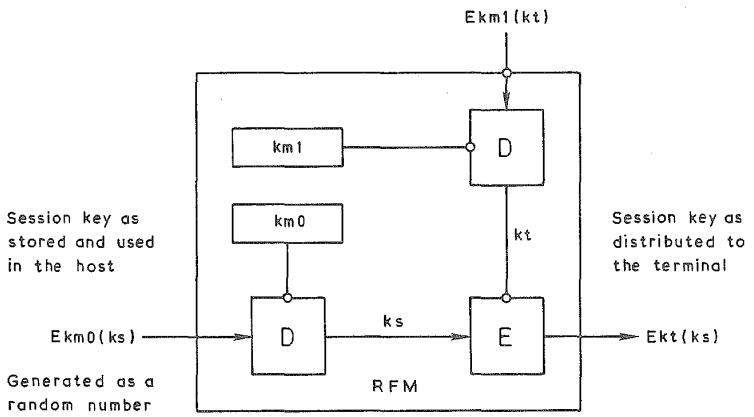


Fig. 6.6 Session key distribution

present 'owner' of the terminal. The enciphered values of the terminal keys, kt_i, can be stored in ordinary, unprotected storage but their use is controlled by the operating system.

It is not allowable to generate a new session key in its plaintext form because that form may only appear inside the TRM. The pseudo-random-number generator is employed to generate a new value of $Ekm0(ks)$, the stored form of the session key. The resulting value of ks will then be a random number, perfectly suitable for its purpose. Generating the enciphered key $Ekt(ks)$ at random is useless because the RFM function works only one way, there is no function available which will take $Ekt(ks)$ and re-encipher it under the master key. It is important that there should not be such a function, otherwise an enemy within the system could obtain the session key by tapping the line while it was being transmitted to the terminal and from it obtain $Ekm0(ks)$ which he could then exploit in the HE or HD functions to encipher or decipher users' data taken from the line. So the one-way property of the RFM function is essential.

In this operation there are two master keys, $km0$ and $km1$. If only one key were used it would be possible, by misuse of the TRM operations, to expose a key in its unenciphered form as shown in Figure 6.7. Suppose that the same master key, $km0$, was used to encipher both ks and kt. Then the quantity $Ekm0(kt)$ could be used as the key for the HD operation and applied to the value $Ekt(ks)$ obtained by tapping the line to the terminal during session key distribution. This trick would produce the session key ks in plaintext form and this could be employed anywhere to decipher users' data obtained from the line. The use of two different master keys prevents this trick.

Generation and loading of a top-level master key is time-consuming, so the need for loading two different master keys must be avoided (and as we shall see later, a third is needed). In the IBM scheme this is achieved by deriving $km1$ from $km0$ simply by inverting a specific set of bits in the value of the key. Although these keys are then closely related, their encipherment properties are completely different and, since both are kept secret, their relationship is not of great importance to the strength of the system.

Since the random value used for starting the key distribution is $Ekm0(ks)$ the value of ks produced is a random block of 64 bits and does not have the correct

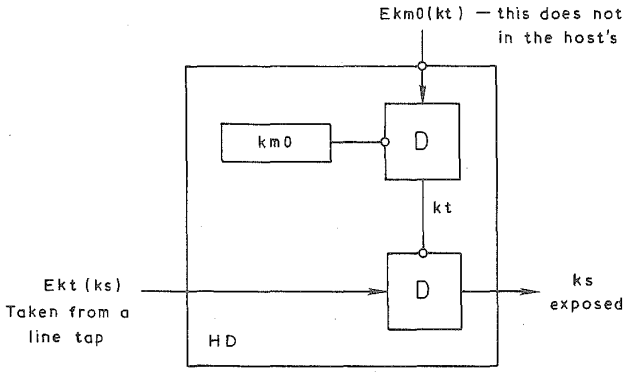


Fig. 6.7 How ks *could be exposed if* km1=km0

parity check properties of a normal DES key. It would be possible to arrange that, when the value *ks* is produced in the TRM, its parity is corrected and thus when it arrives in the terminal and is loaded into the working register it has the correct parity. It seems that IBM did not do this, but allowed the value of *ks* to reach the terminal without its correct parity checks. The DES chip ignores the eight spare bits instead of treating them as parity digits.

Generation and distribution of the terminal key

New session keys are produced frequently but the terminal key is changed less often. When it is changed, the new key value has to be taken to the terminal and loaded into it and for this purpose it must be available in its cleartext form, *kt*. It is therefore an exception to the normal rule that cleartext keys are never exposed. Since it is important that there should be no way of regenerating *kt* at the host after it has been distributed, the only way to organize its distribution is to generate the key in the cleartext form and this presents a new problem, how to form the quantity E*km*1(*kt*) which is the stored value of the terminal key. This operation is, of course, carried out by the operating system and it uses the special quantity E*km*1(*km*1).

A function is provided in the TRM for enciphering under the master key *km*0 and this is known as 'Encipher under Master Key' (EMK). It is a privileged operation which is given the quantity x and returns the quantity E*km*0(x).

The quantity E*km*1(*km*1) is produced when the system is initialized, then it is stored away and kept only for use in the *kt* distribution process. To produce it, *km*0 is used to produce *km*1, a trivial operation, and this is used with the EMK function to generate E*km*0(*km*1). This quantity is used as the key in the HE operation with *km*1 as data, as shown in Figure 6.8, to generate the special quantity needed. After these processes, all stored values of the master keys are destroyed except those in the TRM. A written copy of *km*0 is kept in a safe for disaster recovery. If the system is destroyed, but enciphered files are available in a back-up store, these would be useless unless the master keys could be retrieved.

Using the special quantity E*km*1(*km*1), it is possible to complete the distribution of the terminal key in two stages. First this key is enciphered under *km*0 using

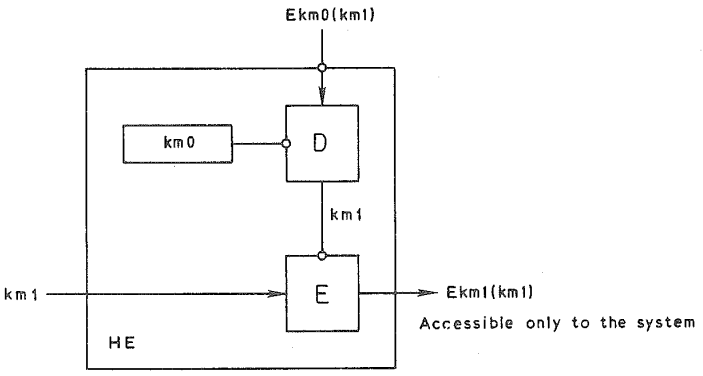


Fig. 6.8 An operation used at system initiation

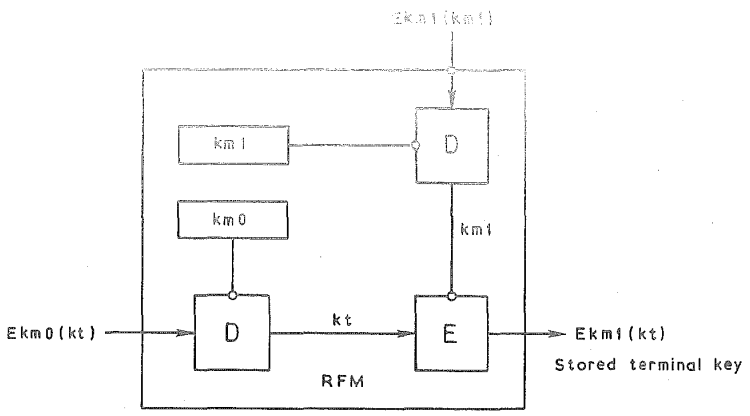


Fig. 6.9 An operation used to prepare a new terminal key, kt

the EMK function, then the RFM function is applied to this quantity, using as key the special quantity E*km*1(*km*1), as shown in Figure 6.9. The result is the quantity E*km*1(*kt*) which can then be kept in store and used for distributing session keys as shown in Figure 6.6.

Notice that the intermediate quantity E*km*0(*kt*) is one which earlier we said should never exist, because it could be misused to expose session keys stolen from the enciphered line. This means that terminal key distribution must be carried out with extreme care, using trusted software and making sure that the intermediate values cannot be accessed by another program.

The principles of file security in the IBM key management scheme

The hierarchy of keys used for file encipherment is very similar to the one used for communication security and is shown in Figure 6.10. The data in the files are enciphered by file keys, *kf*, and file keys are stored under secondary file keys, denoted *kg*, which are held by the owners of the files. A single value of *kg* creates a 'protection domain' which may contain a number of files each having its own data enciphering key. The quantities E*kg*(*kf*) need not be concealed because they cannot be used unless *kg* is available. For example it may be convenient to store

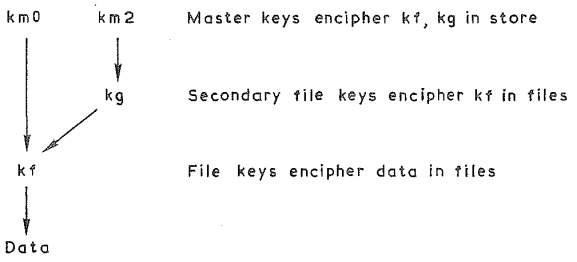


Fig. 6.10 The hierarchy of keys for communication security

the value of $Ekg(kf)$ in the header of the file itself, to make sure that it is always available when the file has to be deciphered. By having these individual file keys, the benefit to an enemy of breaking the cipher to obtain one key is very limited unless the knowledge of the file keys could be used in a further stage of cryptanalysis to obtain the secondary file key *kg*. Handling groups of files together under one key, *kg*, can also enable a group of files to be communicated to another user by giving him the *kg* value. These appear to be the main reasons why the intermediate or secondary file key is included in the hierarchy.

At the top level of the hierarchy are the master keys. The key *km*0 is used for enciphering file keys (as well as session keys) in the host. The key *km*2, derived like *km*1 by a trivial operation on *km*0, enciphers secondary file keys for storage in the host.

The encipherment or decipherment of a file is allowed for any user or process which has access to the secondary file key in its stored form $Ekm2(kg)$. This can be used, as we shall show later, to generate the file data encipherment key *kf* in the form $Ekm0(kf)$ required for employment in the HE and HD functions. These quantities $Ekm0(kf)$ are not stored in the system, they are simply recreated whenever a file is to be deciphered. Thus the control for deciphering a file rests in controlling access to the secondary file key *kg*.

Generating and retrieving a file key

Figure 6.11 shows the TRM function which is used when preparing to encipher a file or retrieving a file key for decipherment. It is called 'Re-encipher to Master Key' (RTM) and is almost a mirror image of the RFM function but using the master key *km*2 instead of *km*1. At the top of the figure, the 'key' input is the secondary file key in the enciphered form $Ekm2(kg)$ in which it is stored in the host. The data quantity entering the RTM function is a newly generated pseudo-random number when the file is first enciphered. This gives rise to a random number *kf* inside the TRM which is the file data encipherment key and is never exposed outside. The result of the RTM function is the file key enciphered under the master

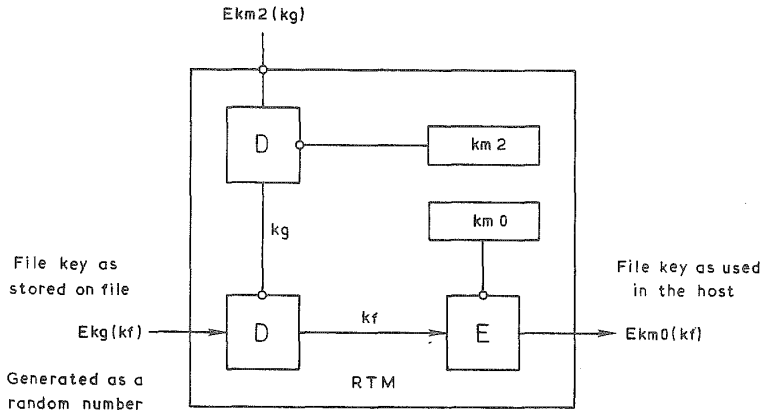


Fig. 6.11 Preparation to encipher or decipher a file

key $km0$ and this is a quantity needed for encipherment or decipherment using the HE or HD function. The quantity $Ekg(kf)$ can be stored along with the file in order to be ready for later encipherment or it can be held by the owner of the file. When decipherment is required, this quantity is passed again through the RTM operation to generate the value $Ekm0(kf)$ used as the 'key' for the HD function.

The clever feature of the design of this system is that it makes the RFM and RTM functions almost the inverse of each other but employs different master keys, $km1$ and $km2$. By employing one of these functions for communication security and the other for file security, it is made impossible to misuse these functions in the wrong context. If $km1$ and $km2$ were identical, RTM could be used to reverse the effect of RFM and thus values of $Ekt(ks)$ stolen from the line would be usable to generate $Ekm0(ks)$ for decipherment.

We have described both the communication security and the file security functions inside a single host. When two hosts are connected together in a communication network, two further functions are required, the transfer of messages under encipherment between hosts and the movement of files from one host to another in an enciphered form. These are provided by extensions to the key management scheme which will now be described.

Transfer of enciphered data between hosts

All we have described so far about the IBM key management scheme concerns a single host. In order to extend this to a number of hosts which are nodes of a network the keys in the system must have an additional index which will be a superscript i or j. For example, the master keys are $km0^i$, $km1^i$ and $km2^i$ for the ith node. A method of transferring data from node i to node j will be described. For this purpose, all that is needed is to be able to establish as many session keys as we need, which are known to both nodes. A session key would be created in node i as $Ekm0^i(ks)$ and then, in the ordinary way, the operation RFM would be used (see Figure 6.6) to encipher this session key under a terminal key. The session key can be sent to another node if there is a suitable 'terminal key' for the purpose. The proper name for this key would be 'secondary communiction key' but to avoid the change of notation we will call it kt^{ij} signifying that it serves for moving session keys from node i to node j. It is important that this 'inter-node terminal key' is directional; it can only function from node i to node j not the other way round.

Figure 6.12 shows how the session key moves from its normal place of storage

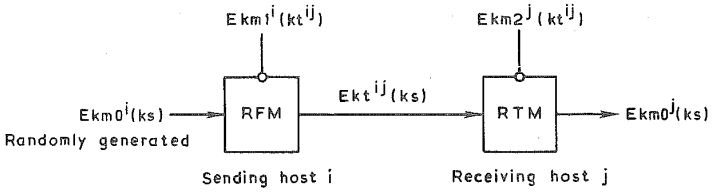


Fig. 6.12 Preparing a session key for inter-host communication

in node i through the intermediate form $Ekt^{ij}(ks)$ to the form in which it is stored in node j. Different users or processes in node i will establish their own session keys in this way, all using the same inter-node key kt^{ij} for the purpose. Then the session keys can be used with the HE and HD functions to enable data to be transferred from one node to the other. In fact the session key can be used to transfer data in either direction. The directionality of the inter-node key relates only to the movement of the session key. For example, kt^{ij} is used when a user or process in node i takes the initiative in transferring the session key to node j.

The operation shown in Figure 6.12 is made possible because the inter-node key is available in each node in a suitable form, using the $km1$ master key in the second node and $km2$ master key in the receiving node. In this respect, the receiving node is anomalous because a terminal key is not usually stored in this fashion. Its existence in node j requires the use of the function RTM to be controlled very carefully. The inter-node key kt^{ij} must be transported between the two nodes outside the communication system, probably in a key carrier, and then established in the appropriate enciphered form for storage at each node. We have already seen how $Ekm1(kt)$ can be established at a node in a privileged operation using the secret number $Ekm1(km1)$. A similar procedure can establish $Ekm2(kt)$ and for this purpose the secret number, $Ekm1(km2)$, is needed. This is generated when the system is initialized, immediately after the master key $km0$ has been loaded, using a procedure like that shown in Figure 6.8. Obviously these functions which operate on clear values of kt^{ij} and establish their appropriate enciphered form at each host must be very carefully controlled, and they are not used very often. When an inter-node key is established between a pair of nodes, it is available for a relatively long period to transmit session keys. A different key, known as kt^{ji}, is established for sending session keys in the opposite direction.

Transfer of enciphered files between hosts

To complete the description of the IBM key management operations, it remains to show how a file can be transferred from node i to node j. This is illustrated in Figure 6.13. The middle part of this figure is very similar to the operation shown in Figure 6.12 but with the session keys replaced by file security keys. The secondary file key kg^{ij} can be regarded as an 'inter-node file transfer key' for sending files from node i to node j. Its establishment uses the procedure described earlier.

The procedure in Figure 6.13 is more elaborate than the communications procedure because the quantities $Ekm0(kf)$ do not normally exist in the system, but are called into being when an operation is to be carried out on a file. The

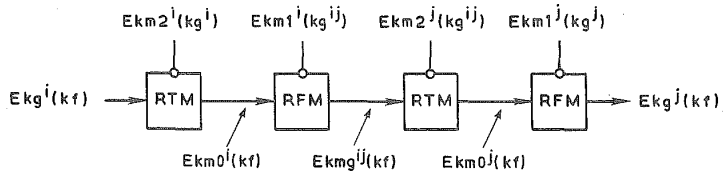


Fig. 6.13 Moving an enciphered file from host i *to host* j

notation here shows secondary file keys such as kg^i, which are the usual secondary file keys belonging to a file-owner, with a designation added to show which node they belong to.

The starting point of any file operation is the enciphered file key such $Ekg^i(kf)$, usually stored along with the file to which it refers. When this quantity goes through the procedure shown in Figure 6.13 it emerges enciphered under a new secondary file key in node j, namely the secondary file key of the user or process which is to receive the file. There is no change to the enciphered data since the file key kf remains unchanged. All that is necessary at the receiving end is to place the new enciphered file key $Ekg^i(kf)$ in the appropriate place in the header of the file.

In this procedure the RFM operation which prepares the file key for leaving node i is unusual, in that secondary file keys are not normally held under encipherment by $km1$ and stored file keys, like $Ekm0(kf)$, cannot normally be transferred to a new secondary file key by the RFM operation. Therefore, it is important to control carefully the use of the RFM function in this context.

The security given by the IBM key management functions comes from limiting the cryptographic operations provided by the TRM to a specific, small set of operations each working with the appropriate set of keys. This restriction prevents the exposure of keys in their clear form or the use of any stolen keys in a different host from the one that they belong to. But the functions used for transferring data or files from one node to another cannot not be made available to every user and these are among the functions which must be controlled by the system in order to maintain security. Other things that must be controlled are the key loading functions and the use of the special numbers $Ekm1(km1)$ and $Ekm1(km2)$. Ultimately, security depends on the control of access to storage and processing functions within the host computers.

6.5. KEY MANAGEMENT WITH TAGGED KEYS

The IBM key management scheme uses encipherment by variants of the master key in order to separate different kinds of keys and ensure that they are used in the correct way. Three variants of the master key are needed so that functions provided for data encipherment and decipherment, session key distribution and file key retrieval cannot be misused for other purposes.

Jones[8] has described a method of handling keys which achieves the same separation of functions with only one master key. The separation of functions in this new method can be applied to an extended key hierarchy without any further complication. The principle employed is to attach a *tag* to each key in its clear text form which identifies that key's function. The idea of tagging words in a computer has often been used, for example to distinguish between capabilities and other words in a capability-based computer. The tag is typically a short extra field attached to the word and preferably one which cannot be separated from the word or altered by non-privileged users of the system. In this section we shall describe the use of the eight 'parity bits', which are attached to the 56-bit DES key, as a tag field. Later, we shall see that the number of tag bits needed is quite

small, but the availability of eight bits helps to simplify the description of the method.

These eight bits were intended to be parity bits and some DES hardware implementations refuse to accept a key unless the parity is correct. There seems no special reason for sum-checking the key, because it is easy to verify that keys have been installed correctly by giving them test patterns. The proposed international standard for DEA1 in its draft form regarded these simply as eight undefined bits without specifying how they were to be used. The ingenious aspect of Jones's scheme is that, since keys are not exposed outside a TRM in their clear form, it is not possible for an enemy, even one with access to the computer, to modify these tags.

To begin our description, assume that each type of key employed in the key management system has its own tag value. The top level master keys are an exception because they never leave the TRM so their method of use is entirely controlled. The other key types are the communication keys, *ks* and *kt*, and the file security keys, *kf* and *kg*. We denote the tags attached to these keys as *Ts*, *Tt*, *Tf* and *Tg* respectively. The key values are concatenated with their tags to form plaintext values which are then stored in enciphered form. A typical example of a key in this system would be $Ekm(ks; Ts)$. The semicolon denotes concatenation where the tag is actually interleaved with the key, occupying the spare bits. Since there is only one master key, we call it *km*.

In order to use the tags as part of the control of key management, each of the operations of the TRM checks the tags of incoming keys and, where appropriate, places the correct tags on outgoing keys. We shall now consider the action of these various functions in turn, beginning with the encipherment and decipherment of data.

There is little change to the functions depicted in Figure 6.5. The incoming session key is tagged in the form $(kg; Ts)$ and the HE and HD functions will only accept a session key if it has the correct tag. With any other value of the tag, the TRM must return a 'reject' signal.

The HE and HD functions are also used, with file keys *kf*, to encipher and decipher files. For this purpose, the HE and HD functions should be prepared to accept keys with either of the tags *Ts* or *Tf*. The essential restriction on these operations is that they should only be used on data, not on keys. So if an enciphered key-enciphering key were presented in place of the enciphered session key at the 'key' input of one of these functions, it would be rejected. Essentially, the *Ts* or *Tf* tags verify that a *data* enciphering key is being used, making it safe to offer the encipherment or decipherment function to a user or process.

Next, consider the RFM and RTM operations. These are operations entirely with keys but the 'key' input of the function takes in an enciphered key-enciphering key whereas the 'data' input takes an enciphered data-enciphering key. Figure 6.14 shows the RFM operation in its new form. It checks the types of the two keys it is handling when their plaintext is exposed inside the TRM. The key which enters at the data input leaves the module (under different encipherment) carrying the value of tag with which it entered. Because of this checking of key types, there is no need for the master key variants.

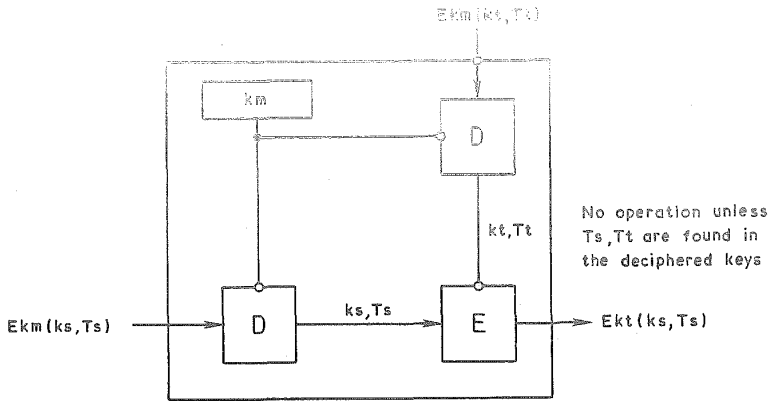


Fig. 6.14 The operation RFM with tagged keys

Generation of new tagged keys

New keys are produced by a random number generator but before they can be used they must be correctly tagged and enciphered. The key enciphering keys, *kt* and *kg*, present no problem because the processes which are allowed to generate these keys have access to the 'encipher with master key' operation which, in this key management scheme, could attach either the *Tt* or *Tg* tag, whichever is asked for. But although a terminal key must be available to the system in clear form for distribution to the terminal, the same is not true of a secondary file key, which is only ever used in the enciphered form outside the TRM. It is, therefore, more secure to insist that a new secondary file key be generated entirely within the TRM, each such key generation starting with a new random number which is not revealed outside the TRM. This prevents an enemy who has penetrated this part of the system from generating identical keys with two different tags so that a key value could be used in both the RFM and RTM operations, something which the system is intended to prevent.

The generation of new session and file keys in the IBM key management scheme is performed in such a way that different operations are used, RFM for the session key and RTM for the file key. The starting point in each case is a random number which represents an enciphered key. With the tagged-key method, a random number will not create a key with the correct tag, so a different method of key generation is needed. One possibility is to include in the operations of TRM, two key generation functions designed for these types of keys. A session key is generated in the form *Ekm*(*ks; Ts*) and can be used for any terminal. In order to generate a file key, the secondary file key *kg* must be provided to the TRM as *Ekm*(*kg; Tg*) and with this input a random key *kf* is chosen and its enciphered form *Ekg*(*kf; Tf*) is supplied from the TRM. Thus each type of key requires a function for generating it in the TRM, designed in such a way that the same key cannot be generated with more than one tag value. Only *kt* keys begin their lives in plaintext form.

Extending the key hierarchy

We have been using a key hierarchy with three levels, at the top the master keys with no tag, at the next level two kinds of 'secondary' keys with different tags and at the lowest level the data enciphering keys (session and file) with their distinctive tags. If we wanted to extend the key hierarchy to four, five or more levels we would need to introduce new tag values and require the various operations of the TRM, in particular the RFM and RTM operations, to recognize new combinations of tags which they will allow.

The proposal by Jones has a different and simpler scheme which allows the key hierarchy to be extended without introducing new tag values. The main distinction which it makes is between data enciphering and key enciphering keys. Only data enciphering keys are allowed to enter the operations HE and HD. The categories *ks* and *kf* that we have distinguished are both included in the data enciphering category. All key enciphering keys belong to a second category, which includes *kt* and *kg*, but these two types of key must be distinguished because we only allow *kt* as a key input to RFM and *kg* as a key input to RTM. This distinction is made by providing two bits in the tag field which determine whether the key can be used for encipherment or decipherment or both. From Figure 6.6 it can be seen that *kt* is an enciphering key and from Figure 6.11 that *kg* is a deciphering key. Therefore three bits form the complete tag; one distinguishes data and key enciphering keys, the others allow encipherment and decipherment respectively, or both. The operations RFM and RTM can be carried out at several levels of an extensive key hierarchy because there are key encipherment keys of several levels. Only data enciphering keys can be used in the operations HE and HD.

The distinction between enciphering and deciphering keys has so far been applied only to key-enciphering keys, leaving the data-enciphering keys such as *ks* and *kf* marked for both encipherment and decipherment. But data-enciphering keys can also be produced for a single role, either encipherment or decipherment. Within a system controlled in this way it should be possible to generate corresponding pairs of enciphering and deciphering keys. This makes possible an asymmetric form of cryptography.

A particular session or file key, with its tag, can be transmitted throughout a network of inter-connected nodes which employ the same tagged key management system. Their security depends on the physical security of all the TRMs in the system and in particular on the safe distribution of the 'inter-node keys' such kg^{ij} and kt^{ij}. The inter-node keys create a problem for the key generation method which we described earlier because the same key is tagged in two different ways in the two nodes it interconnects. We saw earlier that the handling of inter-node keys requires privileged operations. In the tagged scheme it seems to require that the inter-node key, generated as a terminal key for use in RFM at one node, can be converted into a secondary file key for use in RTM operations at the other node. The establishment of inter-node keys uses operations that must be very carefully guarded, whether the IBM method or Jones's variant is used.

5.5. KEY MANAGEMENT STANDARD FOR WHOLESALE BANKING

The international standard ISO 8732 entitled *Banking—Key Management (Wholesale)*[2] describes key management procedures for safeguarding the secret cryptographic keys used to protect banking messages. The scope known as wholesale banking includes messages between banks, between banks and their corporate customers or between banks and a government. This makes it clear that messages of some consequence are being protected and that such applications as home banking and point of sale transactions are not within its scope. The standard is complex and occupies more than one hundred pages. It describes the procedures for manually exchanging keys in order to initiate the system and then, with the aid of these manually exchanged keys, the automatic exchange of the keys which are used to protect the messages and, in particular, to provide message authentication. The major part of the standard is concerned with the automatic methods, giving their procedures, the contents of the messages and the detailed coding.

For someone coming to this standard for the first time, the main problem is the quantity of detail. Our account cannot be a substitute for the standard itself and we therefore do not attempt to cover the detail but describe instead the main principles, leaving out features where this does not prevent its principles being understood. For example, we give little attention here to error messages or error recovery procedures.

The standard is very significant because it is the first internationally agreed principle for key management. Consequently, some of its notation and concepts have already been adopted in other schemes, such as proprietary key management procedures which do not follow ISO 8732 in detail but claim some similarity. Departing from our usual terminology, we shall adopt the word *encryption* in place of *encipherment* to align with the standard, where appropriate.

We begin our account by describing the key hierarchy and the way it uses keys of both double length and single length. The standard was derived from the US standard ANSI X9.17 where the intention, at the time, was that the DES algorithm would be used. For generality, the international standard does not specify that DES shall be used, though the standard ANSI X3.92 (which describes the DES algorithm) is quoted.

Each participating bank or other (corporate or government) organization is expected to have a *key management facility* which includes the cryptographic equipment and provides a secure environment in which the cryptographic functions can be performed and the keys held. The standard does not describe how secure arrangements can be made for the storage of the keys, though their encipherment under some unspecified master key is implied. These are details which are the concern of each user and they are not specified because they do not affect the exchange of messages between the participants.

A short section of the standard, little more than one page together with an example in an annex, deals with the manual distribution of keying material which sets up the basic keys on which the key hierarchy depends.

We continue with a description of the three different environments under which

key distribution can be carried out, according to this standard. For each of these, the messages which are exchanged in normal operation will be described, together with a partial account of the content of these messages.

The standard does not claim to cover all aspects of key management. The banks themselves can decide on the lifetime of keys, the number of keys they use and the provisions they will make to have reserve keys available in case one is compromised. The standard provides for communicating the date and time at which the key becomes effective and for exchanging messages which withdraw a key from service. How these tools will be used by individual banks is left to their judgement. The user can also choose the type of key hierarchy (within limits) and whether to use double or single length keys for some of the functions described. Thus a decision to use the ISO 8732 standard for key management does not complete the agreement between users, who would then have to detail the chosen options and the schedule of keys and key changes.

There are two technical features which are central to this standard, known as *notarization* and *key offset*. Notarization employs the modification of a key using the names of the source and destination of a communication as the modifying data. This key is then used to encipher a subordinate key in a key management message and the receiver of that message can only interpret it by using the correct key modifications by the names employed by the sender. This prevents any intruder diverting a key intended for one purpose into another purpose. There are other ways to do this, but the *notarization* principle is used here. A similar modification of the key encrypting key uses a counter which is incremented throughout the whole life of that key. In this way, the subordinate keys sent with this key encrypting key carry a built-in note of their sequence number and an attempt to reuse an old key would be detected. Notarization and key offset are generally used together.

These two features, and the general complexity of the standard make it questionable whether it should have a wider validity than the scope for which it was intended, but it is invaluable as an example of a fully developed key management standard.

The key hierarchy

Figure 6.15 shows the two alternative key hierarchies employing two or three layers of keys respectively. It also shows the notation for the three types of keys employed.

A key used for enciphering or authenticating data is known as a *data key* (DK) and is the lowest level, layer 1, of each hierarchy. These keys are used for either encipherment or authentication but should not be used for both except when the standard specifies this. Wherever data keys are distributed for use in protecting banking messages, either one or two keys can be sent so that one can be used for encipherment and the other for authentication. The encipherment function of the data keys also includes the encipherment of initializing values for use in the standard modes of operation described in chapter 4. Data keys are always single length keys.

For the distribution of data keys, a *key encrypting key* (KK or KKM) is used. In

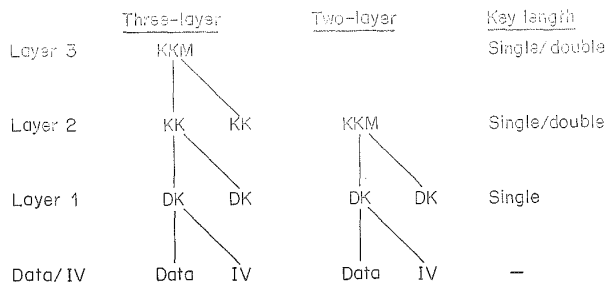


Fig. 6.15 Alternative key hierarchies of ISO 8732, KKM, master (key encrypting) key: KK, key encrypting key, to encipher DKs; DK, data key, to encipher data or IVs or authenticate messages

the two-layer hierarchy a master key is used, which has been established by manual key distribution for this purpose. In the three-layer hierarchy any KK which is used to encipher a data key for transmission can itself be distributed using a master key encrypting key (KKM).

Since it is widely accepted that the 56-bit key size of the DES algorithm weakens it, under certain circumstances, there is sometimes a need for multiple encryption to obtain an effectively longer cryptographic key. Where keys are in use for a long time, the opportunity for an attacker to perform a key search is greater, therefore this standard envisages the possible use of double-length keys at the higher levels of the hierarchy. The method by which encipherment is performed with these double-length keys is described in the next section.

The use of double-length keys is mandatory only for master keys which are used for key management between a participant and one of the two kinds of key management centre (either a key distribution centre or a key translation centre). Apart from this requirement, key encrypting keys (KK or KKM) can be either single or double length but a single length key must not be used to encipher a double length key. The standard does not give any criterion for the use of double length keys but we would expect a double length key to be used wherever the life of the key exceeds about one month. The presence of a double length key is signified by the notation *KK or *KKM.

Encipherment and decipherment with double length keys

Two single length keys each of 64-bit size can be concatenated to form a double length key. Thus a double length key *KK is considered as being made up from two single length keys KKl and KKr where the notation signifies the left and right-hand halves of the concatenated double length key. The encipherment and decipherment processes using these keys are shown in Figure 6.16. In each case three cryptographic operations are employed. For double-length encipherment these are encipherment, followed by decipherment, then by encipherment. Both encipherments use the left-hand part of the key and the intervening decipherment uses the right-hand part. This triple processing was chosen because the simpler double process could be weakened by a 'meet-in-the-middle' attack, employing a large amount of storage in order to reduce the computational task of key searching.

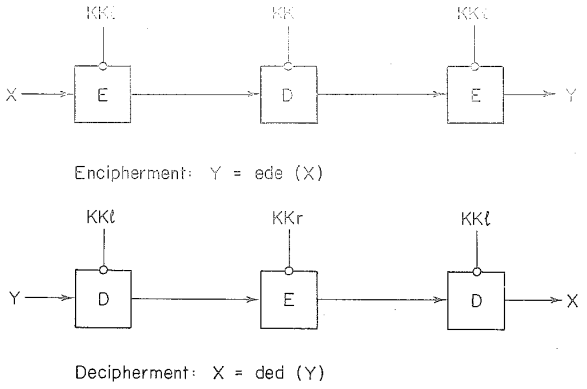


Fig. 6.16 Triple encipherment with a double length key

The corresponding decipherment function shown in the figure employs decipherment followed by encipherment then decipherment with the same key sequence. The notation for encipherment using this process is *ede* and correspondingly for decipherment *ded*.

It is not entirely certain that the use of double length keys provides an effective key length which is double that of a single length key, but the structure of DES and the experimental evidence that it does not form a group supports the view that multiple encipherment is more secure. There is, however, no need to use multiple encipherment unless the threat of a key-search attack is significant.

Key distribution environments and messages

The simplest environment is that two banks wish to establish keys for the exchange of messages and they already have in common a key encrypting key, either a master key (KKM) or a key at layer two of a three-layer hierarchy (KK). Their aim may be to distribute a KK for the three-layer hierarchy or to distribute data keys or perhaps to do both. No other party is involved. This is the *point-to-point* environment.

To avoid the need for banks to have master keys or key encrypting keys in common with all their correspondents, it is possible to use a key management centre which has an established cryptographic relationship with both parties to the communication. Then, using the key already established they can obtain the necessary material from the key management centre to establish contact. There are two kinds of key management centre, one of which generates keys, called a *key distribution centre* and the other which accepts a key from one party, re-enciphers it and returns it in a form suitable for communication with the other, called a *key translation centre*. Thus there are three environments in the standard, known as *point-to-point, key distribution centre* and *key translation centre*. For each of these there is a procedure for carying out the key exchange which we shall describe with reference to Figures 6.17, 6.18 and 6.19 respectively.

In describing these procedures we do not detail all the messages which can be used. In particular, any message may be in error and receive a response which

is an *error service message* (ESM). In the content of the key translation centre there is an *error recovery service message* (ERS) which reports errors to the centre and requests resynchronization and reinitiation. In this account, we shall confine ourselves to the messages used in normal operation, when no errors occur.

The coding of these messages employs the character set of International Standard ISO 646, often known as the ASCII code. The messages are coded in a way which makes them readable when printed on the page. There is an intention to introduce another version of the standard with binary coding of messages. In the coding, both the message type and the tag for each field of the message is a set of three capital letters. Within the message, the various fields are tagged to indicate their significance and separated by space characters. Additional format characters can be inserted to improve the ease of reading the message. For our purpose, we shall not define the format and coding in detail.

Point-to-point distribution of keys

When two banks A and B already have a key encrypting key in common they can use it to exchange other keys of the form KK or KD. The messages they use are shown in Figure 6.17. For example, bank A can send a *key service message* (KSM) to B containing a key or keys enciphered under the already established key. Bank B will return a *response service message* (RSM) and this completes the exchange. But suppose that bank B wishes to establish keys in this way and does not have the ability to generate keys. It is then possible for bank B to send the optional *request service initiation* message (RSI) shown by a broken line in the figure, which stimulates bank A to begin the procedure already described.

All messages in this and the other protocols we shall describe contain three particular fields, the first defines the message type, for example a key service message would have a field MCL/KSM where MCL is the tag for the message type. The other fields which all messages contain are the identities of the originator and receiver of the message. We shall not repeat these fields because they occur in all messages.

All messages also contain either an authenticator (MAC) or else an error detection code (EDC). Whenever a message is of cryptographic importance and a suitable authenticating key is available, a MAC is included. In the point-to-point procedure, the KSM and its response RSM have MAC values but the RSI message does not. This is because authentication always uses a data key and the purpose of this procedure is to establish a data key (or keys) so the existence of such a key cannot be assumed. The KSM and RSM messages both include a MAC using the data key which has been transmitted in the key service message.

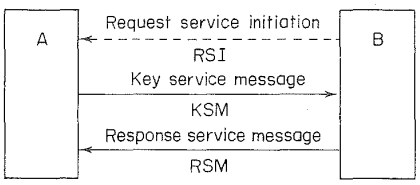


Fig. 6.17 Basic messages in point-to-point key distribution

The error detection code (EDC) is formed in the same way as a MAC but uses a fixed cryptographic key value 0 1 2 3 4 5 6 7 8 9 A B C D E F in hex notation.

The key service message can contain a key encrypting key, enciphered under a master key already in existence but this is optional and will not occur in a two-layer hierarchy. It always contains at least one data key and can contain two if it does not also have a key encrypting key. If there are two data keys present, the key used for authentication of the key service message is formed from their exclusive OR. The KSM may also contain an initializing value (IV) enciphered with a data key, the second one if there are two present. The RSM also contains the date and time at which the data keys become effective.

For each key encrypting key shared by the two banks, a counter is maintained which starts at the value 1 when this key is first used to encipher KD values. It is incremented for each subsequent use and the counter value is sent in the message. The receiver checks the counter and responds with an error service message if it is wrong.

Notarization is an optional feature in the procedure for point-to-point key distribution. If it is used the field NOS/ with no parameters is included in the message. In this case the KK is subject to a transformation by notarization and key offset before it is used to encipher the data key.

The response message contains no more than the fields contained in all messages, mentioned above and the MAC. The MAC is formed with the same key that was used for the key service message, therefore the receiving bank must decipher the KD value or values before using them to check the received authenticator and generate an authenticator on the response. Reception of a correct RSM by bank A, when its MAC value has been checked, verifies that the correct KD was received and hence a correct KK, if one was sent. There is no corresponding check on the correct receipt of the IV.

In addition to the messages shown in Figure 6.17 either bank may send the other a disconnect service message (DSM) which, in addition to its standard fields includes the identity of the key to be discontinued. The DSM also includes the identity of a key used for authenticating it and a MAC value. The response to the disconnect service message is an RSM containing the identity of the discontinued key and a MAC using the same key as for the DSM. Note that the content of a response service message varies according to the type of message to which it responds.

Key distribution centre

If bank A does not have a data key in common with bank B it may be able to establish such a key through the agency of a key distribution centre (CKD). For this purpose, bank A first communicates with the distribution centre and obtains from it a new data key in two forms. One of these it can interpret using a KK which it holds in common with the distribution centre. The other it passes on to the bank B which itself can interpret this using another KK which it has in common with the same distribution centre. The scheme of messages used is shown in Figure 6.18.

There is an optional request service initiation (RSI) from bank B to bank A, in

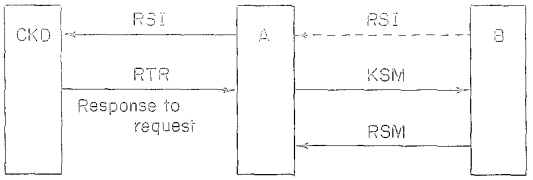


Fig. 6.18 Basic messages using a key distribution centre (CKD)

case bank B wishes to stimulate bank A into carrying out this protocol. Normally, bank A would initiate the protocol with an RSI sent to the distribution centre. In this procedure, only the RSI messages use an error detection code, all the others employ a MAC. The RSI from bank A indicates the service required and the identity of the 'ultimate recipient' which is bank B.

The service provided by the key distribution centre is the supply of one or two data keys and optionally an IV enciphered under a data key, the second one if there are two. Note that a key distribution centre does not supply key encryption keys. This contrasts with the key translation centre which is able to translate key encryption keys if a superior master key encryption key exists.

The response to request (RTR) is returned by the distribution centre and contains the usual three fields identifying the message type, the originator and receiver and also a MAC using as key a data key or, if there are two, a key produced by their exclusive OR. In addition, the RTR includes the identity of bank B, the data key or keys in two forms, an optional IV and the date and time at which the data keys become effective.

The key distribution centre maintains a counter which is incremented each time it uses the key encrypting key, either the one for bank A or the one for bank B. The two values of these counters (CTA and CTB) are sent in the message. Concerning the data key or keys, these are provided in two forms called KD and KSU, the first intended for bank A and the second for bank B. Each is notarized, that is to say the KK used is modified according to the source and destination of the key and it is also offset with the value of the counter CTA or CTB.

Bank A receives the RTR, extracts the KD value or values from their notarized encipherment and verifies the MAC. It can then generate a MAC for the key service message which it sends to bank B containing the keys in the form KTU, together with the IV, date and time of effectiveness of the keys and the counter for bank B (CTB). Bank B can then extract the KD value and verify the authenticator, after which it returns the RSM. Both the KSM and the RSM contain the identity of the key distribution centre. Finally, bank A verifies the MAC received in the RSM.

Key translation centre

A more powerful facility is provided by the key translation service offered by a key translation centre (CKT). In this scheme, bank A generates the keys it needs and passes them to the translation centre for conversion to encipherment under the keys held in common between that centre and bank B. The message which initiates this procedure, request for service (RFS) contains the key values that will

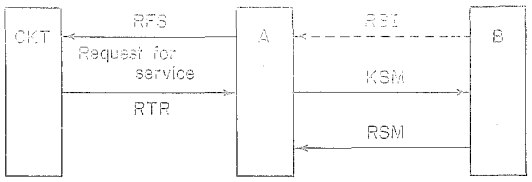


Fig. 6.19 Basic messages using a key translation centre (CKT)

eventually arrive at bank B, see Figure 6.19. As before, the whole procedure may optionally be initiated by bank B using an RSI message to prompt bank A into starting the procedure. Apart from the RSI, all the other four messages in this procedure are protected by authentication using a MAC based on a data key which is transmitted (in enciphered form) with the message for that purpose. Though the names of the messages are similar in these various procedures, the content is different according to the procedure in which they take part. The handling of the counters CTA and CTB is similar to that of the key distribution procedure, so also are the identifying data, such as the identity of bank B in RFS and RTR and the identity of the key translation centre in KSM and RSM. Effectively, each important message holds the identity of all three parties.

The key translation centre translates either one KK or one or two data keys. It never translates keys of both types. Keys to be translated are sent enciphered under existing superior keys but not notarized. If a KK is sent for translation a data key is also sent, enciphered under this KK, but is not intended for translation, it is merely for calculation of the MAC values.

The response, RTR, contains the translated keys, either a KK or a KD or a pair of KDs. They are notarized for transmission between the translation centre and bank B, and offset using the counter CTB. The value of CTA is sent in the RFS and returned in the RTR. Bank A receives the RTR and verifies its authenticator using the data key it originally sent. It then transmits the notarized keys to bank B, together with the CTB value received from the translation service.

If a key encrypting key has been translated, new data keys are generated by bank A to include in the KSM, enciphered under the new KK. At least one such data key is needed to provide the means for authentication of the KSM and its response. Two data keys may be sent in the KSM and may subsequently be used between the two banks as working data keys. In addition, bank A may generate an IV and send this enciphered under a data key, the second key if there are two of them. Bank A also decides the effective date and time (EDK) for the new keys and includes that in the message. Bank B deciphers first the KK, if any, then the data key or keys before verifying the authenticator and returning the RSM, which contains only identifying data and a MAC. Finally bank A verifies the authenticator of the RSM.

The full procedure for operation of the key translation centre includes error service messages (ESM) and error recovery service messages (ERS) which help to report and recover from loss of synchronism of the counters. As with the other two environments any one of the parties can send a disconnect service message (DSM) to terminate the operation of a specified key and receive a response to verify that this has happened.

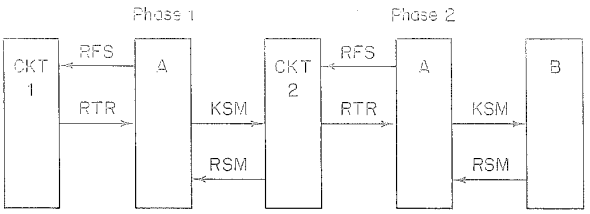


Fig. 6.20 Using two key translation centres in turn

Consecutive use of two key translation centres

Suppose that bank A has a key relationship with translation centre CKT1. By key relationship we mean that the two parties share a key encryption key. Suppose also that CKT1 has a keying relationship with another key translation centre CKT2. Bank A wishes to communicate with bank B for which the normal key translation centre is CKT2. There is a scheme which enables bank A to communicate with bank B by using the procedure shown in Figure 6.19 twice over. The method is shown in Figure 6.20 in which, for convenience, bank A is shown twice.

In the first stage, bank A uses the normal procedure with key translation centre 1 to establish a key encrypting key in common with CKT2. Having done this, bank A then interacts with CKT2, and since CKT2 has a key in common with bank B, it is able to translate bank A's key for delivery in a KSM message to bank B. At the end of these two stages, banks A and B have a key encrypting key in common. They can then exchange data keys by the point-to-point procedure and use these data keys to protect their messages.

It is clear that this principle could be extended to translate keys through a number of translation centres which are linked by mutual KK values. A group of translation centres could organize themselves to provide a comprehensive service in a number of ways, for example appointing one of them to translate keys between any of the others. These possibilities are not described in the published standard, which quotes the arrangement of Figure 6.20 as an application of the key translation procedure.

This extension to more than one centre is not possible with the key distribution centres because they can distribute only data keys and therefore cannot establish a new KK relationship.

Key notarization and offset

Key notarization and offset are applied to any keys supplied by a key distribution centre and to the translated keys from a translation centre. These keys are supplied under encipherment by a superior key and the notarization and offset procedures consist of modifying the superior key before using it for encipherment. Notarization means that the key is modified according to the source and destination of the message, offset means that the key is modified according to a counter value, which provides a sequence number for the successive keys

enciphered with this superior key. The purpose of notarization and offset is to prevent the replay of a message carrying encrypted keys and to prevent the substitution of one message for another. This same purpose could have been achieved in other ways but in this standard the notarization and offset techniques are mandatory in the key distribution and key translation functions. In the case of point-to-point key distribution, use of notarization is optional.

Figure 6.21 shows the calculation involved in key notarization and offset. We begin with a key encrypting key of double length which has two parts, KKl and KKr. The application to single-length KK will be described later. The identities of the source and destination of the key are given by double-length fields entitled *TO* and *FM*. The two halves of TO are called TO1 and TO2 and similarly the two halves of FM are called FM1 and FM2. The figure shows how a combination of encipherment and exclusive OR is used to combine these values to form a double-length value called the notary seal (NS). This is the quantity which is used to modify the key *KK before it is used for encipherment.

The figure also shows the use of key offset, in which the counter value CT is combined by exclusive OR with each part of the notary seal before this is used to modify the double-length KK which is to be used for enciphering the key. The result of these operations is the double-length value *KN which is used in the triple encipherment process *ede* to encipher the key for transmission. Normally this transmitted key will be a data key, but it could be a key encrypting key enciphered by a master key encrypting key. A similar procedure can be used with single-length KK values employing the convention that KKr equals KKl. A single-length notary seal value is generated by combining the 32-bit halves of the double-length value as shown at the right of the figure. The subsequent operations are applied to a single-length value to generate a single-length modified key KN used for the encipherment of a transmitted key.

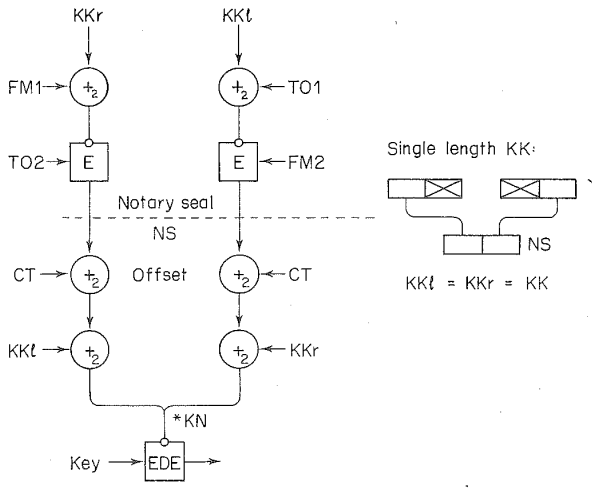


Fig. 6.21 Notarization and key offset

6.7. ALTERNATIVES FOR KEY MANAGEMENT

We have described (but not fully) two elaborate key management schemes designed with special circumstances in mind. In each case it is possible to see how alternative and sometimes simpler schemes could have been developed with much the same properties. It seems that the science of key management is still developing.

General purpose cipher algorithms and modes of operation are now an established fact but key management is tailored to individual applications. The need for this adaption will become even clearer when we describe the special circumstances of point-of-sale EFT in chapter 10. For all kinds of applications, key management standards will become necessary and they will have some features in common and many points of divergence. Progress in this field still seems unclear. The international standard ISO 8732 includes some valuable concepts and notation but should not be taken as a point of departure for all future key management standards.

There is great potential for the use of public key cryptography in key management. This technology is introduced in chapter 8 and the application of these asymmetric algorithms to digital signature is described in chapter 9. Both are valuable techniques for improving the security of key management. Standardization in their key management role has not begun but ISO 8732 envisages that there will be, in due course, provision for the use of asymmetric algorithms for key distribution. The most immediate use is to employ public key cryptography to distribute keys for symmetric algorithms as typified by DES. The security requirements for the safe use of public key cryptography are described in chapter 8 and digital signatures, as described in chapter 9, provide a superior method of checking the integrity of key distribution messages and resolving disputes concerning messages sent and received.

Key management in electronic funds transfer systems (EFT) has been developed both with symmetric algorithms and public key cryptography. Chapter 10 contains descriptions of key management of both kinds.

REFERENCES

1. Ehrsam, W.F., Matyas, S.M., Meyer, C.H. and Tuchman, W.L. 'A cryptographic key management scheme for implementing the Data Encryption Standard', *IBM Systems J.*, **17**, No. 2, 106–125, May 1978.
2. *Banking–Key Management (Wholesale)*, International Standard ISO 8732, International Organization for Standardization, Geneva, 1988.
3. Matyas, S.M. and Meyer, C.H. 'Generation, distribution and installation of cryptographic keys', *IBM Systems J.*, **17**, No. 2, 126–137, May 1978.
4. Heinrich, F. *The network security center: a system level approach to computer network security*, NBS Special Publication 500–21, January 1978.
5. Needham, R.M. and Schroeder, M.D. 'Using encryption for authentication in large networks of computers', *Comm. ACM*, **21**, No. 12, 993–999, December 1978.
6. Popek, G.J. and Kline, C.S. 'Encryption and secure computer networks', *Computing Surveys*, **11**, No. 4, 331–356, December 1979.
7. Denning, D.E. and Sacco, G.M. 'Timestamps in key distribution protocols', *Comm. ACM*, **24**, No. 8, 533–536, August 1981.
8. Jones, R.W., 'Some techniques for handling encipherment keys', *ICL Technical J.*, **3**, No. 2, 175–188, November 1982.

Chapter 7 IDENTITY VERIFICATION

7.1. INTRODUCTION

The security of a system often depends on identifying correctly the person at a terminal. A familiar example is a bank's 'Automatic Teller Machine' which gives cash to account holders that it can identify. Direct person-computer interaction is good for efficiency, but it opens possibilities for fraud by a *masquerade* — the adoption of a false identity. Access to computers and entry to buildings should be controlled according to the identity of the person, but like many human skills that we take for granted (such as language and vision) recognizing a person is surprisingly hard for a computer.

Suppose that a bank has a million customers. If it is to recognize them by a personal identification number (PIN), that number must have at least six decimals and if instead it recognizes by a signature, the measurement of the signature must separate each person's signature from all the others. The bigger the population, the harder it is to make a positive identification. Fortunately for us, this is not the problem usually presented. We assume that the terminal user wants to be known. We can ask him his identity and it can be entered on a keyboard or from a card he carries. We can use this claimed identity to access a file giving the reference 'measures' of his signature or his reference PIN. If the given data match the reference, we accept him. Suppose he is an impostor who does not have the correct signature or password, then there is a small probability that he may be lucky, for example a four-digit password, well-chosen, makes this probability 10^{-4}. With the right precautions, this can be adequate because nobody would risk being caught for such a small chance of success. Nevertheless, the probability of a successful masquerade must be chosen according to the amount at stake — an instant pay-off of £10 million is worth a 10^{-4} chance try if the risk of detection is small enough. The important characteristic of identity verification is that the precision of the method need not be increased as the population increases. It depends rather on the ratio of the potential illegal gain to the 'loss' value of being caught. Practical systems do not *identify* the individual but do verify the identity he claims to have.

Methods of personal identity verification can be divided into four broad categories, though particular systems can contain elements of more than one category. These categories employ, (a) something known by the person, (b) something possessed by the person, (c) a physical characteristic of the person, (d) the result of an involuntary action of the person.

A password and a passport are examples of categories (a) and (b). Categories (c) and (d) are not always distinguished — we can quote a fingerprint as an

example of a physical characteristic and a signature as the result of involuntary action, since each movement is not individually controlled.

Any successful identification system must, in addition to being economically viable and adequately secure, be acceptable to the system users. An inconvenient or objectionable method is likely to provoke users into finding ways to circumvent the procedures, making the task of any potential intruder much easier.

7.2. IDENTITY VERIFICATION BY SOMETHING KNOWN

Passwords

The best-known example of identity verification by something known is the password. Passwords have been used very widely in identity verification in an enormous range of contexts, from Ali Baba's opening of the magic cave, through military applications, to protocols for accessing computer systems. Much attention has been given to methods of generating and managing passwords. A standard for PIN management was published by ANSI in 1982.[1]

We can divide passwords into several categories, (a) group passwords which are common to all users on a system, (b) passwords which are unique to individual users, (c) passwords which are not unique to individual users, but serve to confirm a claimed identity, (d) passwords which change every time a system is accessed. Each of these types of password has seen some use in our context.

Group passwords are sometimes used as part of the log-on procedure for a service being accessed from a terminal. However, being common to all users, they are likely to become known outside the circle for which they are intended. Disclosure usually happens through a breach of security such as writing the password on the wall by a terminal, a regrettably widespread practice. In any system where more than the minimal security is needed, group passwords should be ruled out.

Passwords unique to each user potentially provide a much higher level of security. If illegal access takes place, then a system log should be able to show under what password the access took place. Control of this kind can detect the use of a loaned password, disclosed by the user to some acquaintance. On the other hand, careless security in storing or handling a password may lead to its unintentional disclosure; this could also be said to be the fault of the person to whom the password has been allocated. Good security practice entails memorizing the password and not keeping a printed copy. If unique passwords are implemented, then a separate user identifier need not necessarily be used, though an added confirmation of identity can be generated. In a user population which is very large, as, for example, the users of a bank's automatic teller terminals, use of unique passwords is impracticable — such a password would have to be too long for easy memorizing.

This implies a third kind of password which is becoming increasingly common — the non-unique password. Here a relatively short password, of, say, four decimal digits, is allocated to each user. Identification of the user depends on a much longer number which the user is not required to memorize — it may be held on the magnetic stripe of a card, for example. The fact that the same password

is allocated to a number of different users presents no problem if the password allocation bears either no relationship or no easily determined relationship to the user identity. In an on-line system using non-unique passwords, a list can be held centrally which gives user identities and their corresponding passwords; identity verification depends on looking up the presented identity and password on this list. In an off-line system, the password must be related to the identity token, for example the magnetic-striped card, in such a way that the off-line terminal can interpret it whilst interpretation by the user (or intruder) is too difficult; this implies some kind of encipherment.

There is a fourth kind of password — one which changes every time the system is accessed. To operate such a system it is necessary to prepare a list of passwords and give a copy of the list to the user. At each access one password is used and then becomes invalid for future access. This prevents an intruder from acquiring a current password by tapping a communication line because the traffic on the line gives no indication whatsoever of the next acceptable password. Clearly the system is stronger than those systems which use the same password(s) repeatedly and is useful where the highest security is required. However, there are certain disadvantages; the user must protect his printed list of passwords and the password list requires storage and protection at the central installation. The amount of password material depends on the frequency of access. One-time passwords of this type are in use in the Society for Worldwide Interbank Financial Telecommunications (S.W.I.F.T.) network of the international banking community described in chapter 10. For greater security the S.W.I.F.T. password tabulations are prepared in two halves, so that each list contains only half passwords. The two halves are sent separately to users, reducing the chance of complete passwords being intercepted.

It is of course necessary to design the password handling by the system in such a way that it is protected against disclosure. For this reason, any password that is transmitted from a terminal to a central installation should be protected during transit by encipherment, otherwise a passive line tap will suffice to betray it to an intruder. Equally, the response from the central installation to the terminal accepting the validity of the identity and password must be protected by encipherment. If this is not done, then the accepting response can be reorganized and reproduced on the line by an intruder with an active line tap; if the terminal were an ATM this would allow an intruder to defraud the system without actually knowing any password. Furthermore, some variability must be included in both request and reply (for example, serial number or time/date), otherwise a standard request or reply would lead to a standard enciphered format. All the intruder need then do would be to record the standard message and replay it whenever he chose; a replayed request would deceive the central system or a replayed reply would deceive the ATM.

We have referred to lists of identities and passwords held at central installations. Such lists could easily become available to operational staff or systems programmers; on this account it is important not to store passwords in clear form. The usual solution to this problem is to encipher the passwords in some way; the original idea for using enciphered password lists is due to Needham, cited by Wilkes.[2] If the encipherment is carried out with an algorithm such as the Data Encryption Standard (DES), then the risk is transferred to disclosure of the

encipherment key which must be given suitable protection. At the time when the DES algorithm had been specified, but not implemented in hardware, the algorithm was used by the Bell Laboratories for protection of passwords in their Unix system.[3]

The original suggestion by Needham made use of a one-way cipher function; this is a function that is easy to compute in one direction, but very difficult to invert. This type of function is considered in detail in chapter 8. The Bell password protection system made a novel use of the DES algorithm as a one-way function in a programmed implementation. Instead of using a secret key to encipher the password list, the first eight characters of each individual password were used as a DES key to encipher a constant; the DES algorithm was iterated twenty-five times and the resulting 64-bit field repacked to become a string of eleven printable characters. The result of this operation was stored in the password list. The strength of this system was considered adequate until fast hardware implementations of the DES became available; it was feared that fast DES operation would allow exhaustive search for passwords, using the enciphered forms from the password list. For this reason the programmed algorithm was altered so that the internal E permutation (see chapter 3) became dependent on a random number chosen (and entered into the password file) at the time of registration of each password. Thus standard DES devices could no longer be used for a systematic attack on the password security.

Figure 7.1 shows the flow of variables in password checking with a one-way function.

The Bell study considers the time taken to search for a password. On the assumption that an intruder has the opportunity to program a system to try a sequence of passwords without being detected and thrown off the system, it is possible to predict how long it would take to test all possible passwords against a given identity. On a PDP11/70 it took 1.25 ms to encrypt and test each potential password and compare the result with the password file. For a sequence of four lower case letters the time taken to check all possible passwords was 10 min; for a four-character sequence selected from ninety-five printable characters the time was 28 h; for a sequence of five characters chosen from the set of sixty-two alphanumeric characters the time was 318 h. Evidently a password consisting of

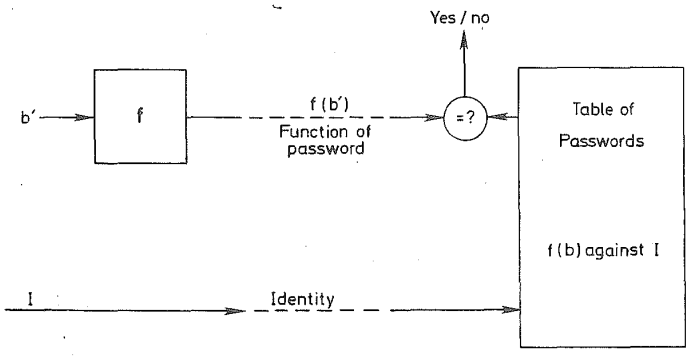


Fig. 7.1 Password checking with a one-way function

four lower case letters would be pitifully inadequate. A judgement must be made that weighs password length (with problems of memorizing longer sequences) against the value of the protected entity.

Choice of password presents some interesting problems. It has been shown that users asked to choose passwords without any advice or constraints are prone to select character sequences that are easily predictable; this is because they choose sequences that they consider to be easily memorable. The Bell report casts a fascinating light upon user habits when no constraints were placed on password choice. Of 3 289 freely chosen passwords examined the character strings used were as follows: 15 were a single American Standard Code for Information Interchange (ASCII) character, 72 were two ASCII characters, 464 were three ASCII characters, 477 were four alphanumeric characters, 706 were five letters, all upper case or all lower case, and 605 were six letters, all lower case.

They also discovered that favourite choices of passwords were dictionary words spelt backwards, first names, surnames, street names, city names, car registration numbers, room numbers, social security numbers, telephone numbers, etc.; to an intruder these are all worth testing before considering less-likely character sequences. Of the 3 289 passwords the Bell authors collected, 492 fell into one or other of these categories. Added to the list of short sequences already given, this meant that 2 831 passwords, or 86 per cent of the sample, were either too easily predictable or too easily searched for.

It seems from this evidence that the practice of leaving choice of password to the user can result in the use of unsatisfactory passwords. An alternative is to have the choice of password made by the system. For this purpose a series of random characters can be generated, but the user will find it difficult to memorize totally random character streams, even if the string length is limited to eight characters, say.

Memorability is known to be enhanced if the character stream can be chosen so as to be pronounceable, because users find syllables more easily memorable than characters. Therefore a generator that is constrained to obey certain built-in pronunciation rules may be able to produce passwords which are difficult to predict but not impossibly difficult to remember. For example, 'SCRAMBOO', 'BRIGMERL', 'FLIGMATH' and 'LIDFRANG' are all nonsensical, but easily pronounceable and therefore memorable. The total number of different eight-letter character strings is about 2.1×10^{11}. One particular password generation scheme[4] produced a measure of the fraction of all possible eight-character strings that were pronounceable; this was found to be about 0.027, giving a total population of pronounceable eight-letter strings as 5.54×10^9. A large commercial English dictionary would contain as many as 2.5×10^5 different words of all lengths, so the random number generator used in this mode produces vastly more potential passwords than are found in a dictionary; hence such passwords are far less predictable.

The task of secure transmission of passwords from a central system to the users deserves attention. Unless the passwords are transported securely, their value is nullified. Encipherment is no solution because verifying the correct identity of the receiver presents the same problem that the password is intended to solve. It seems advisable to use some other means of transport such as the letter post for password transmission. The banks use this technique when sending PINs to

their customers. In order to prevent interception of the password during transit, it is recorded inside a sandwich envelope at the time of generation by a computer. A character printer imprints the password on the sandwich; as there is no ribbon in the printer the password does not appear on the outside of the envelope; inside there is a sheet of carbonless copy paper, so the password gets printed internally, but is only readable when the envelope is torn open. If the envelope is interfered with in transit, then this is obvious to the intended recipient, should he eventually receive it; any apparent interference should lead to the rejection of the password. In banking practice the PIN is used in conjunction with a plastic card which is sent separately from the PIN; in order to receive a PIN the authorized user must acknowledge safe receipt of the card to the bank. An intruder must obtain both card and PIN before being able to secure any profit from his action. A modest degree of security is obtained by this technique, but it could not be applied where high security is called for. In the latter case it may be necessary to use a trusted courier service to transport passwords or hand them over personally to the user.

There is a school of thought that believes that the password should not be known to the issuing organization. One system[5] that allows a personal choice of password unknown to the system makes use of a one-way function of a combination of personal account number and password; the password is chosen by the user on the occasion of a visit to the bank and keyed on a keyboard unseen by any bank official, combined with the personal account number and then stored in the bank system. To identify the user at a subsequent visit to the bank, the same one-way function is used. The password is not known by the bank at any time; if the user forgets the password, then another personal visit to the bank is necessary in order to choose and activate a new password. Since the password is not known to bank employees, manipulation of the account cannot take place by normal system channels, though this does not rule out improper access by privileged system operators, for example using the command to change to a new password.

When the screen of a VDU requests 'Enter your password' and the user taps his password on the keys, he is taking a risk. The system has not yet authenticated itself to the user so he might be seeing a masquerade of the actual system, designed only for the purpose of capturing his password. This trick has been widely used as a student prank. When a guard holding a gun asks for your password you have no choice, it is an unequal situation. Perhaps this led to the habit of using one-way passwords in computer systems. But if each party needs to authenticate the other, there should be an exchange of passwords.

When system and user exchange passwords there is a dilemma. One party gives his password before the other and he has no guarantee at that moment that the other party is not a masquerader. This problem can be overcome in a way that is illustrated in Figure 7.2. Here there are two parties, Ann and Bill, holding passwords P and Q respectively. Each knows the other's password for checking purposes. They do not directly exchange passwords but use a one-way function to reply to a challenge from the other party. For example Ann challenges Bill by sending the random variable x_1 and Bill replies with a one-way function y_1 of this value, namely

$$y_1 = O(Q, x_1)$$

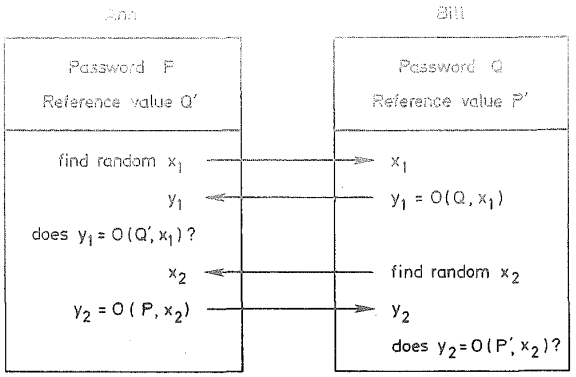


Fig. 7.2 Two-way password checking

The one-way function must be such that even when x_1 and y_1 are both known it is not possible in practice to determine Q. Although Ann cannot determine Q from this reply she can check her own value of Q against the value used by Bill in applying the same one-way function to the value x_1 which she created. If this check is successful she accepts Bill as the genuine owner of the password Q.

In a similar way Bill issues the challenge x_2 and receives the response y_2 in the form

$$y_2 = O(P, x_2).$$

This can also be checked so that Bill now has determined that Ann knows the password P.

This method fails to be secure if the range of the passwords is insufficient. For example, in banking practice the personal identification number is often only four decimal digits long. The system described above using such a short password is very weak because the 10 000 possible values can be checked and, for one of them, the response will match the response received. The challenger can determine the password of the other party.

It might seem that this problem can be overcome by padding the password with a random number and revealing the padding only after the challenges and responses have been exchanged. For example, let the 16-bit password Q be concatenated with a 40-bit random number R_1 to form a 56-bit variable and then let Bill respond to the challenge x_1 from Ann with

$$y_1 = O(Q||R_1, x_1).$$

In this expression, $Q||R_1$ is the 56-bit concatenation of these two variables.

Suppose that each party has carried out this challenge and response, then it remains for them to exchange the values of R_1 and R_2 so that they can verify the responses and thus authenticate each other. But the problem has not been solved because the dilemma 'Who should give the password first?' has been converted into the dilemma 'Who should first reveal the random padding?' As soon as one party reveals his random number, the other party can carry out a password search to find his password.

Variable passwords based on a one-way function

The variable password gives improved security against line-tapping but requires the distribution of new sets of passwords to replace those used up. An ingenious method of using a one-way function has been proposed which provides a sequence of passwords by distributing only one number. It uses a one-way function $y = O(x)$ and iterates this function many times. We use the notation that $O^n(x)$ is the result of operating the one-way function n times, beginning with the variable x. For example

$$O^3(x) = O(O(O(x))).$$

To establish the password sequence, Ann takes a random variable x and forms the value y_0 to send to Bill:

$$y_0 = O^n(x).$$

This transfer need not be made in secret since the seed value x cannot be discovered.

For the first password Ann sends the value

$$y_1 = O^{n-1}(x).$$

Bill can easily check this since $O(y_1) = y_0$ and he has already received y_0 as the initializing value.

The sequence of passwords continues in this way. The ith password is

$$y_i = O^{n-i}(x).$$

Each password can be checked by its relation with the previous password, which has itself been checked. The sequence terminates at the nth password which is x itself.

If passwords are lost or an error occurs, provided that Bill has a recent authentic password value P, a new offered password can be checked against this value by repeated operation with O to cover the gap between the offered password and P. In this way the system can re-synchronize after a failure such as might be caused by a line error.

Questionnaires

Another method of identity verification makes use of information known to the authorized user but unlikely to be known to others. On first introduction to the system the user is asked a series of questions relating to such apparently irrelevant things as name of school headmaster, colour of grandmother's eyes, name of favourite author or football team, etc., rather than to the more usual components of a curriculum vitae, such as place and date of birth, maiden name of wife, etc. Not all the questions need be answered, provided that enough are answered. The user might add his own questions and answers, if the system allows this feature. The selected items are such that the user is likely to find them easy to remember, but an intruder is likely to find them difficult to discover. If desired, the answers may be totally fictional, in which case an intruder may find them even more difficult to discover. On later visits to the system for access, the user is asked a selection of questions to which the system has stored the answers.

Correct replies allow the desired access. Greater security can be obtained if, during an access session, the system asks further questions from time to time; this would detect a session taken over in some way by an intruder.

The questionnaire system of access control has the advantage of using easily memorable information, but it does involve a rather lengthy exchange between the user and the system, which may be found tedious and inconvenient by some, especially if repeated. It is hardly likely to be acceptable as the means of controlling access to a bank automatic teller terminal, for example. Furthermore, if the value protected in the computer by the access control system is great enough, the determined intruder can take the trouble to research the background of the authorized user sufficiently to provide the questionnaire answers. Therefore this technique is not likely to be used in systems requiring a high level of security.

7.3. IDENTITY VERIFICATION BY A TOKEN

In the widest sense, access control by means of something possessed by the authorized user is very familiar. Nearly all of us possess keys to operate locks in cars, filing cabinets, house doors etc. Computer system access may be made dependent upon possession of a key which fits a lock installed in the computer terminal. The first action of a user is to enable the terminal using his key, after which normal log-on procedure may follow. For adequate security the lock must be of a complicated (and therefore expensive) type, preferably 'unpickable'; there must be a wide range of possible key variations (known in the locksmith trade as 'differs') to fit locks of the particular class in use, so that the intruder cannot make use of a full set in his possession, trying each until one is found that will fit. Unfortunately, if a key is lost or loaned to someone, it is usually fairly easy to copy it, even though the lock mechanism may be complicated. Locksmiths have agreements that keys of particular types are not copied without proper authorization, but even this will not deter the determined intruder. The copied key may be restored to its owner without the latter even being aware of its being missing. A lock and key is a useful way to restrict access to one or two selected people, such as those authorized to load cryptographic keys, but it is not of general use for access control. The problem of 'copyability' is typical of all token-based identity verification.

Magnetic stripe cards

A token which is becoming as common as the physical key is the plastic card with a magnetic stripe. Such cards are increasingly being used for identification purposes, for automatic teller machine (ATM) and point-of-sale operations, for credit validation, for access control to secure sites etc. An International Organization for Standardization (ISO) standard[6] gives dimensions for card and stripe; other proposed standards define the use to which each of the recording tracks on the stripe shall be put, including the relevant data formats. The general card dimensions and layout are indicated in Figure 7.3. Amongst other data the user identity is stored on the magnetic stripe. Generally speaking the magnetic stripe card is used in conjunction with a type of password called a personal identification number (PIN). In off-line systems the PIN must also be stored on the stripe

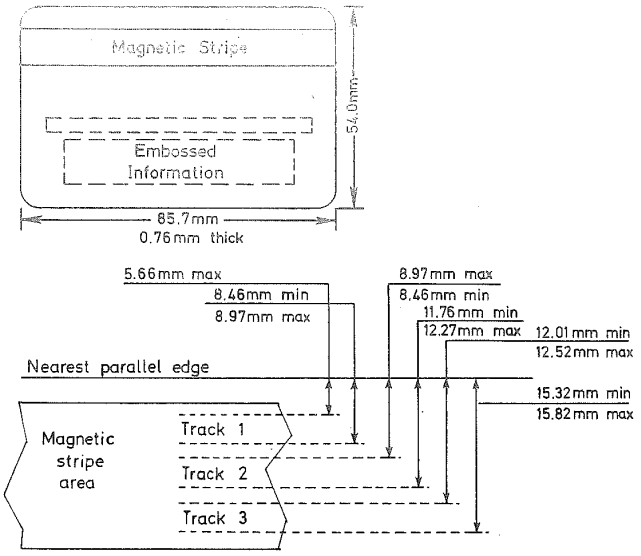


Fig. 7.3 Layout of magnetic-stripe cards

in enciphered form. The recognition equipment reads the identity, deciphers the PIN and compares the latter with the PIN keyed in by the user. In on-line systems the reference PIN need not be carried on the card, but can be stored in the central system; the claimed identity is notified to the central system together with the offered PIN; if the latter agrees with the reference PIN, then access is permitted.

As we have remarked already in our discussion of passwords, the user must be educated to take proper care of his PIN. It is not unknown for users to write their PIN on the card itself in order not to forget it; this is clearly asking for trouble. There are also slightly more subtle forms of attack. Cards often carry the user name imprinted so that an intruder can easily discover the user identity corresponding to a stolen card. If the card is an ATM card, the enemy telephones the user, claims to be representing his bank and asks for the PIN 'in order that we may restore your card to our system'. If the card does not carry the user identity, the latter may still be discovered because the bank national number and user account numbers appear; these identifiers appear together with users' identity on cheques.

Ideally, PINs should be memorized by their users and not written down in diary or notebook. However, this may be asking too much of the average user, especially if a large number of cards is carried. It is said that the average number of credit cards carried by American businessmen is eleven, each of which may have its distinct PIN.

Unfortunately it is very easy to copy the usual kind of magnetic stripe card. Cards can be manufactured, without excessive difficulty, to resemble true cards sufficiently well as to deceive most people having to deal with them. One widely used method of hindering the production of forged embossed cards is the

hologram covering one or two digits of the embossed number; any attempt to change the embossing destroys the hologram. The contents of the magnetic stripe can be transferred to another stripe without expensive equipment. Therefore it is important to consider means of making cards much more difficult to counterfeit. Methods devised for this purpose tend to concentrate on hindering the reproduction of the data on the magnetic stripe. Many 'security features' have been invented to improve the security of magnetic recordings on cards; two will be described here.

Watermark tape

The Emidata/Malco system[7] establishes a permanent, non-erasable, magnetic encoding in the structure of the tape. The permanent recording is made during the manufacture of the tape by exposing it to a sequence of magnetic fields whilst the magnetic particles (held in suspension in a resinous lacquer) are still wet. The particles in most tapes (for example gamma ferric oxide) are in the form of long thin needles (acicular); to produce the watermark pattern the particles are first aligned by application of a steady magnetic field at 45° to the longitudinal axis of the tape. A pattern of current pulses is then applied to a special recording head as the tape passes beneath it; when the current is present the particle orientation is changed to the other 45° orientation. The tape then passes through a drier and the orientation of the acicular particles is fixed permanently. Figure 7.4 shows the arrangement of particles. Emidata use the trade name 'Watermark' for tape prepared in this way.

To check the permanent record on a stripe with this structure a special reader is required; in this reader the tape is exposed first to a constant magnetic field and then the fixed recording is read by a head suitably oriented. Because the tape is first exposed to the constant field before reading, this operation is restricted to track 0 of the stripe, a track which is not present on most current magnetic stripes. On tracks 1 to 3 recording and reading is carried out by heads with normal orientation; the normal recording and reading is not affected by the presence of the underlying structure in the magnetic tape. It is, therefore, possible to read a permanent pattern of ones and zeros from track 0 which is not alterable by a forger; this recording pattern can be used to provide valid identification of the stripe and therefore of the card. It would be virtually impossible for a forger to reconstruct the underlying recording in such a way as to deceive a Watermark reader; any attempt to record the 45° pattern on normal tape would be frustrated by the initial exposure of track 0 of the tape to the constant field in the Watermark reader. The layout of the various tracks is shown in Figure 7.5.

The watermark contains 50 to 100 bits of data which are fixed and it is not easy to code these to suit the user of each card. In many cases, random numbers are

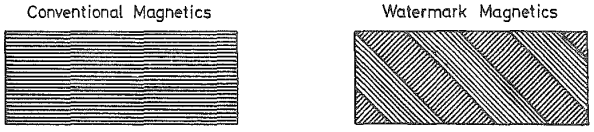


Fig. 7.4 Arrangement of 'Watermark' particles

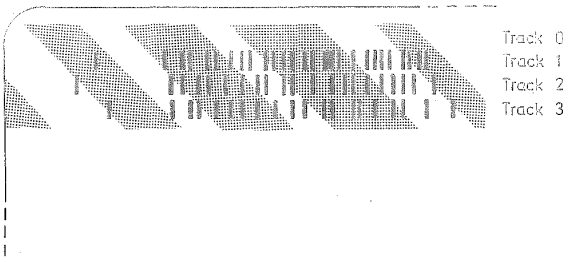


Fig. 7.5 Layout of 'Watermark' tape

used. If W is the watermark number and D the data on tracks 1 and 3 to be protected, an enciphered check field, $C = Ek(W, D)$ can be stored on the magnetic stripe as well as D. Thus the reader can verify this relationship if the secret key, k, is used in its checking device. An enemy cannot compute C and therefore cannot use one card to hold another's data, since their W values are different. The data D can be changed in a device where k is available by altering C to match the new data, but generally it is fixed data that are protected, such as the account number and PIN validation data. It is a weakness that k must be available at all places where the watermark is checked (the checking need not be done at the terminal, however, if a link to a central place is available). The use of public key ciphers (see chapter 8) makes it possible to check using only non-secret data, but increases the data storage requirement a great deal.

The whole strength of the system depends upon the difficulty faced by a forger in counterfeiting watermarked tape. The manufacturing process is such that a forger would find it very difficult to do this, since the process requires significant capital investment and unique technical skills.

Sandwich tape

An alternative to Watermark tape is the tape sandwich in which low- and high-coercivity recording media are bonded together with the low-coercivity tape nearest the recording heads. In order to write on such a tape so that the record is made in both layers a head is required which will produce a high-intensity field. In the tape reader, the tape first passes through an erase field such that any recording in the low-coercivity layer is removed, but the recording in the high-coercivity layer is not disturbed. The 'permanent' recording is then read in the usual way. Figure 7.6 illustrates the structure of the tape and the nature of the recording.

Security of a system using sandwich tape is based on two premises; first, if a forger uses ordinary tape in the manufacture of a forged card, this will be frustrated by exposure of the card to the erase field in the reader; second, if a forger seeks to use the sandwich tape on a stolen card and make a recording of his own, he will find it difficult to create the high-intensity field required for writing into the high-coercivity layer. There is no doubt that use of ordinary magnetic tape is frustrated by this technique, but it seems that high-intensity recording heads might not be too difficult to obtain. There is, therefore, some doubt whether the sandwich tape provides an adequate solution to the problem of making the recordings on plastic cards unforgeable.

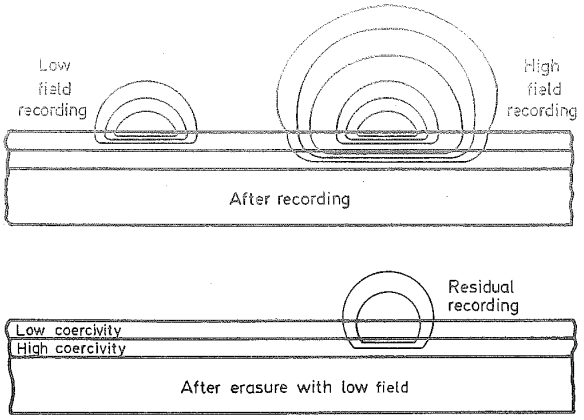


Fig. 7.6 Structure of 'sandwich' tape

Active cards

A recent development is the 'active' or 'smart' card, in which microprocessor or store chips are embedded in the plastic card. This enables the card to extend its storage capacity beyond what is possible on a magnetic stripe (about 250 bytes) and to engage in some data processing. The problem until recently has been the difficulty of introduction of the chip into the card. It has been seen as a requirement that cards of this type shall be of the same dimensions as the standard card with magnetic stripe (8.57 cm × 5.40 cm × 0.76 mm); it is also desirable that the chip card shall be as durable as the standard card. Developments in chip technology have now advanced to the point where the chip is small enough to fit into the required space, thickness being the critical dimension; new methods of mounting and connecting to the chip allow sufficient flexibility. Contacts to the circuitry are via plated areas on the card surface; power supply will almost certainly be applied via this route so that the card is active when it is in a terminal device.

At present the type of chip inserted in a card can be a processor with about 8 Kbytes of store in electrically alterable ROM; on the other hand, other cards contain only memory. This of necessity limits the kind of operation that may be carried out.

Information stored on the card may be semi-permanent, stored in ROM, in which case it may have the same sort of function as that usually carried at present on magnetic stripes — user identification, etc. Illegal alteration of the ROM contents would almost certainly be much more difficult than alteration of a magnetic stripe; therefore, security of such a card is enhanced. Such a card can play a role in identity verification similar to that which we have already outlined, though the additional security and amount of storage might allow more-sophisticated systems.

One application of a card with stored data is for viewing pay-television. If the television transmission is sent out enciphered, such that a decipherment key is required in order to make it usable, then the decipherment key, changed every

20 s, say, can be broadcast with the television signal but encoded under a key (a key enciphering key (KEK)) which is held on the plastic card of the viewer. Decipherment of the transmitted key takes place in the card, so that the KEK is not taken out of the card. Possession of a valid card enables the successive transmission encipherment keys to be decoded and the television signal to be made intelligible and used in the receiver. From time to time the KEK is changed, so that a card becomes out of date and a new card must be obtained.

If the card has encipherment capability, this can be invoked in an identity verification procedure. The secret key is held on the card and each card issued has a different key according to the user to whom it has been issued. Identity verification can then be made to depend on the card making the correct response to a challenge issued by the verification terminal; the terminal issues a test pattern which the card encrypts and returns to the terminal; the latter checks the response against the encipherment key which should be contained in the card, correct response implying presence of the right key. This system has the advantage that the response depends on the challenge and, therefore, an attempt by a potential intruder to tap the channel between card and verification terminal would yield little of use in making an attack upon the system.

If a system can be devised in which sufficient rewritable store can be established on a plastic card, then many applications become possible. Bank customers may be supplied with cards which allow them to make purchases in shops at point-of-sale terminals; the card is initially 'charged' at the bank with value (debited to the customer's account) which is decremented at the point-of-sale terminal, the amount of the decrement being transferred as a credit to a corresponding data store held by the retailer. When the customer's card is exhausted, it is returned to the bank for 're-charging'. The retailer periodically takes his stored data to the bank for credit to his account. This type of system is described in chapter 10.

The physical security of data on a 'smart card' is improved by storing all the sensitive data on the same chip that holds the processing and encipherment facilities. Active or passive attack can be made difficult, but there is not yet available an independent assessment of the level of physical security obtained. The thin plastic card is not the only format for an 'intelligent token' containing storage or processing chips. Tokens could be physically similar to a small calculator or like a physical key that turns in a 'lock' which is really a reading terminal. It is possible that greater physical security can be provided in enclosures a little larger than a card.

Another development uses plastic cards which contain stores which are read once and then erased. One such storage medium is the hologram. The kind of application where a system of this type may be used is the purchase of a card with a set initial value, to be used against the provision of a service, for example, transport, telephone or pay-television; a recent example is the 'Cardphone' service of British Telecom, which uses a holographic medium. As the service is used, so the storage on the card is progressively read and erased, bit by bit; finally the card has no more value. When the card is exhausted it is thrown away and another card purchased. To the supplier of the service it is important that the erasure of the card value is permanent, otherwise illegal 'refreshment' of the card may take place with loss of revenue. It is also important that it is excessively difficult