1-1-1999

Development of a model for smart card based access control in multi-user, multi-resource, multi-level access systems

David Shaw
Edith Cowan University

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

Development of a Model

for

Smart Card Based Access Control

in

Multi-User, Multi-Resource, Multi-Level Access Systems

by

DAVID SHAW BAppSc (Comp & InfoSc)

Faculty of Science, Technology and Engineering
Edith Cowan University
10 April 1999

A Thesis Submitted in Partial Fulfilment of the Requirements for
the Award of
Master of Science (Computer Science)
at the
School of Computing and Information Science
Faculty of Communications, Health and Science

Date of Submission: April 10, 1999

Abstract

The primary focus of this research is an examination of the issues involved in the granting of access in an environment characterised by multiple users, multiple resources and multiple levels of access permission. Increasing levels of complexity in automotive systems provides opportunities for improving the integration and efficiency of the services provided to the operator.

The vehicle lease / hire environment provided a basis for evaluating conditional access to distributed, mobile assets where the principal medium for operating in this environment is the Smart Card. The application of Smart Cards to existing vehicle management systems requires control of access to motor vehicles, control of vehicle operating parameters and secure storage of operating information.

The issues addressed include examination of the characteristics of the operating environment, development of a model and design, simulation and evaluation of a multiple application Smart Card. The functions provided by the card include identification and authentication, secure hash and encryption functions which may be applied, in general, to a wide range of access problems.

Evaluation of the algorithms implemented indicate that the Smart Card design may be provably secure under single use conditions and conditionally secure under multiple use conditions. The simulation of the card design provided data to support further research and shows the design is practical and able to be implemented on current Smart Card types.

Declaration

I certify that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously published or written by any other person except where due reference is made in the text.

Sipnature.

Dave Shaw

Date.....14DEC99.....................................

Acknowledgement

Any work placed in the name of the author relies on the support and assistance of many people who must be acknowledged. I will apologise beforehand should I inadvertently omit the recognition owing to those who have helped with their time and advice. This acknowledgment is with gratitude for their efforts throughout the time taken to complete this work.

Most importantly; my wife, Alison, for her humour, help and consideration during the days spent with computer and book for without her support this work would never have been attempted.

My supervisor, Associate Professor W Kuczborski, who provided support when the enthusiasm lagged and the labour seemed never-ending. Professor AC Watson for his guidance in the early stages of the work.

Advice from professionals in many fields provided insights and starting points to explore some of the many facets, so appealing, in the work. The list, while not conclusive, must include Professor U Maurer of the Swiss Federal Institute of Technology, Professor J Pieprzyk of the University of Wollongong, Professor V Cordonnier of the University of Lille and Professor E Dawson of the Queensland University of Technology.

The support and encouragement from colleagues, Dr SP Maj, Ms S Jones, Dr T Vuori, Mr M Wetton and Dr TJ O'Neill must be recognised as well as the support staff, especially Ms. C. Dixon.

Where the original theme was conceived as part of an undergraduate final year project, appreciation of the work of the staff at the University of South Australia is due. Specifically, Mr. BS Bowden, the project supervisor, Mr R Frick and Mr. K White.

Finally, for the friends who have endured my fascination with the topic and helped where they could with support and encouragement, my appreciation to Mr. GW Byrne, Mr SF Neeson, Mr GN Rae, Mr GM Sherrington and Mr EWF Shaw.

To conclude, I acknowledge without hesitation the vast debt owed to the people mentioned and the ongoing work of many people not explicitly mentioned here. To sum up my experience in completing this work I can only echo the sentiments of another, more well known :

'If I have seen further it is by standing on the shoulders of giants.'
Isaac Newton.

1. Glossary and Explanation of Terms.

1.1. Mathematical Terms.

A	:	Number of cells in an array
B	:	a binary number
C	:	number of unique combinations from sampling R
D	:	a decimal number
d	:	the index generated by the selection algorithm
d_n	:	average distance travelled by the drunk in Drunkard's Walk
E	:	encryption method
Ei	:	i'th bit of encryption method E
F	:	A User of a Resource administered by the proposed system
G	:	A Resource administered by the proposed system
H	:	Level of Access to a Resource administered by the proposed system
K	:	Key sequence
Ki	:	i'th bit of Key Sequence
k	:	an arbitrary constant
L	:	length of bit sequence necessary in Test T.
M	:	Movement of pixel Square in pixels
M	:	Message of arbitrary length
Mi	:	i'th bit in message
N	:	Number of steps taken in Drunkard's Walk
N	:	integer length of randomiser R
n	:	integer length of sample taken from randomiser R
O	:	number of bits specifying the composite sequence ordering, $\Re$
R	:	Randomiser. random data sequence from which samples are produced
R	:	binary random string of length L= KT, publicly-accessible.
S	:	Set of integers specifying absolute of discard values for selection algorithm
S	:	number of selections taken from randomiser R
SP	:	Start Point, initial entry into randomiser
T	:	Test-T, method of specifying the output of a random sequence generator
u	:	number of possible rotations of a basic sequence
V	:	size of a pixel square side
W	:	Keystream of length N
X	:	Plain text of length N
x	:	number of bits per side of pixel square
x!	:	specification of x * x-1 * ...3 * 2 * 1.a factorial number
Y	:	Cryptogram of length N obtained by adding X and W bitwise modulo 2

Z	:	Secret Key
$\Re$	:	rotation sequence
$\aleph$	:	Composite Sequence
$\oplus$	:	Addition Modulo 2
ω	:	constant equal to 2π
λ	:	length of step in Drunkard's Walk calculations

1.2. Descriptive terms

Absolute Method : method of specifying bits in a randomiser by the index of the bits. (See Discard Method)

Access control : restricting access to resources to privileged entities

Ancillary Systems : separate systems which provide aid or functionality to the operator of a vehicle eg. Air-conditioning, central locking, power windows

Anonymity : concealing the identity of an entity involved in some process.

Authorisation : conveyance, to another entity, of official sanction to do or be something

Basic Sequence : a sequence of symbols used to create a larger sequence by rotation and concatenation

Block Cipher : an encryption method that processes blocks of data, typically 64 bits, according to a specific algorithm

CAN - Controller Area Network : a bus architecture using micro-controllers to administer the control and reporting of information in a mobile environment.

Certification : endorsement of information by a trusted entity

Confirmation : acknowledgement that services have been provided

Complete Composite Sequence : a composite sequence that has one of each possible rotation of the basic sequence.

Composite sequence : a sequence created by rotating and concatenating at least one basic sequence where a rotated sequence may be used more than once.

Complete Ordered Composite Sequence : a Complete Composite Sequence that has, ordered from 0 to n-1, each rotated Basic Sequence.

Complex Composite Sequence : a composite sequence created by rotating and concatenating multiple basic sequences of differing lengths.

Complete Complex Composite Sequence : a complete complex composite sequence of all the rotations of all the basic sequences.

Data Integrity : ensuring information has not been altered by unauthorised or unknown means

DeBruijn Sequence : sequence such as '00011101' containing the set of overlapping three bit sequences '000', '001', '011', '111', '110', '101', '010', '100' comprising a complete binary representation from 0 to 2^3-1.

Derived data : data drawn from other types of data

DES - Data Encryption Standard :an iterative block cipher method.

Discard Method : method of specifying bits in a randomiser indirectly by specifying the number of bits to discard between selections.

ECU - Engine Control Unit : used to control the running of an engine, generally includes digital technology

EEPROM - Electrically Erasable Programmable Read Only Memory : non-volatile memory used in IC cards.

EMU - Engine Management Unit : same as ECU

Entity authentication or identification : corroboration of the identity of an entity (e.g. a person, a computer terminal, a credit card etc.)

External Systems : systems external to the vehicle which interact with or provide information to the vehicle. Eg Global Positioning Systems

Function Generator : a mathematical concept, a block level description of a process where the input and output are linked by the defined function.

ICE : Internal Combustion Engine, where the combustion of the fuel is in enclosed space to maximise fuel efficiency

ISO : International Standards Organisation

Law of Large Numbers : the characteristics of a group of samples will be same as the characteristics of the population

MD5 : a secure hash function from R. Rivest that has been available since 1992.

Message authentication : corroborating the source of information; also known as data origin authentication

Non-Overlapping Sequences : taking bits from a sequence and using them once only. For example, with sequence 1001 the non-overlapping sequences are 10 and 01

Non-repudiation : preventing the denial of previous commitments or actions

Overlapping Sequences : taking bits from a sequence and using them more than once. For example, with sequence 1001 the overlapping sequences would be 10, 00 and 01.

Ownership : a means to provide an entity with the legal right to use or transfer a resource to others.

Privacy or Confidentiality : keeping information secret from all but those who are authorised to see it.

'Random By Definition' : a term used to mean a sample that is considered to be random within the canonical or generally accepted meaning. This implies that the sequence is able to be tested to show a uniform random distribution but is in all senses ideal and not practical.

Randomiser : a sequence of random or pseudo-random symbols

Raw Data : data taken from the sensors before processing

Revocation : retraction of certification or authorisation.

Round Off Error : A round-off error is where some number close to the exact value is used instead of the actual number, for example, the use of 3.1416 instead of 3.141592654 or other representations of greater precision.

Run Length Encoding : a method of reducing storage space for information by encoding the data for length. Suited to systems where the data does not change often.

Secure Hash Function : a method of reducing a larger block of data to a unique, shorter sequence which is linked to the text by a function.

Selection : an unordered multiset with repetition

Selection Key : a set of integers that consist of a Start Point and a Selection Set

Selection Set : a sequence of integers used to specify symbols in the randomiser

Sensor data : data taken from the sensors after processing

Signature : a means to bind information to an entity

Start Point : the initial position of the pointer in a circular buffer

Stream Cipher : an encryption method that processes one bit at a time, usually by addition modulo 2 with a key stream bit.

Timestamping : recording the time of creation or existence of information

Unique Sequence : any selection without replacement of bits from the randomiser

Validation : a means to provide timeliness of authorisation to use or manipulate information or resources.

VMU - Vehicle Management Unit : an electronic control system for a vehicle that manages the engine and ancillary systems and may provide interaction with external systems.

Witnessing : verifying the creation or existence of information by an entity other than the creator

2. Introduction

2.1. Historical Perspective of Motor Vehicle Development

Motor vehicle development is dependent on component technologies such as electricity and metal-working. The potential for powered vehicles was demonstrated by Trevithick's steam carriage in 1801 though smaller, lighter and more efficient engines for industrial and agricultural equipment provided the main impetus for development.

Increased efficiency results when the fuel is burnt inside a closed space to reduce thermal losses. Internal combustion engines use liquid fuels which combine substantial energy with reduced storage space. The 'Four Stroke' internal combustion engine is most common where, in two revolutions of the crankshaft, fuel and air are drawn into a cylinder, compressed, ignited and the combustion products expelled to produce further rotation of the crankshaft.

Further improvement in performance required increasingly complicated mechanical and electrical engine control systems which are now being replaced by electronic components that control multiple systems.

2.2. Basic Structure of Internal Combustion Engines

The internal combustion engine requires a source of fuel and fuel ignition coupled with a suitable mechanical environment for sustained operation. Petrol engine ignition systems rely on mechanically driven components to generate and control the intensity, timing and duration of a spark to initiate fuel combustion in the closed cylinder.

The engine-driven distributor provides a means of initiating, adjusting and distributing a spark to each cylinder in turn. The spark is created by altering the flow of electric current through a 'step-up' transformer which converts a high current, low voltage into a low current, high voltage which is discharged across an air gap to ignite the vaporised fuel. Distributor mechanisms provide fine variation on the timing and duration of the spark during different operating conditions.

The carburettor is a fuel metering device where the vaporising of the fuel for delivery to the cylinders takes place. The vaporisation rate depends on the amount of air drawn through the carburettor which is controlled by the position of the throttle and the size and number of the metering jets. The air demand is produced by the rotation of the crankshaft which uses the cylinders to pump air through the engine.

The basic components of the engine include a reliable electrical ignition system, a carburettor for fuel and speed control and power delivery mechanisms such as the crankshaft, gear box and differential. In recent years, fuel control has developed to the point where complex emission control and engine management systems are in use.

2.3. Electrical and Electronic Developments

Electronics has provided means to improve the efficiency and performance of the internal combustion engine. The weakest point of the distributor, the switch contacts that interrupt the current flow, have been replaced with a transistor and the carburettor has been replaced with electronically controlled fuel injection.

Continuing automotive vehicle development has produced the Electronic Fuel Injection (EFI) system which uses a small computer to control engine performance. One of the major variants was developed by Robert Bosch GmbH. (Gregory, 1990, p8)

The Bosch system uses electronic control systems to provide efficient operation from data provided by the engine sensors. The sensors measure air pressure, temperature and flow, throttle position, engine operating temperature, exhaust gas oxygen content and engine rotation. (Gregory, 1990)

Continued development has made the fuel-injected, electronically-controlled engine very common in new cars. Electronic control of accessories such as air-conditioning and access control has been added to the electronic suite in the vehicle. External systems may also interact with the on-board systems.

Consequently, the controller is required to integrate functions such as authorising and monitoring vehicle use and interaction with electronic traffic control and management systems. This integration of on-board systems with external systems clearly sets the trend for integrated systems development.

A range of topics will be considered in developing the model for the system :

Access Control that identifies users and operating conditions; remote authorisation; security against theft from the vehicle and unauthorised use.

❑ Event logging and short and long term data storage to report on vehicle operation.

❑ Storage of abnormal conditions information such as oil or brake pressure failure.

❑ Administrative functions such as servicing scheduling, fuel usage and other functions.

It is considered that provision for some of these options can be effected by setting limits and options within the software fitment to the Engine Management Unit.

2.4. Research Justification

Initial research into computer security methods indicated the possibilities of applying a developmental algorithm to a range of secure applications. Further discussion with Professor VM Cordonnier and Professor AC Watson highlighted the opportunities for research in multi-user, multi-resource, multi-level access systems typified by the administration of a fleet of lease vehicles.

Many vehicles have integral electronic components and interaction with external and ancillary systems is primarily managed by computers. It is noted that the major issues in distributed, mobile computing are identification and authentication of data and participants in the system operations.

These issues are fundamental to the underlying legal and commercial structures in administering vehicle lease and substantial opportunities for further research may be determined. Additionally, management of the vehicle during operation, secure data processing and transferring of permission to use a resource must be addressed.

Smart Cards provide computing facilities in a portable form with substantial inbuilt security features.

The existing prototype environment is the result of continuing work on Smart Card control of motor vehicles already being undertaken by Professor Watson. The decision to implement the developmental algorithm on a Smart Card links this work with the wider lease environment.

This research will support ongoing research in distributed computing and electronic commerce and provide data about using Smart Cards in the administration of mobile, distributed resources.

2.5. Research Methodology

The prototype environment is constrained by continuing work in Smart Card control of motor vehicle performance being undertaken at Edith Cowan University. Consequently, the provision of a working Smart Card environment reduces the need to construction of prototypes.

Much broad spectrum research is already taking place in automotive, communication and computer systems engineering. Exploration of the lease environment will identify the key issues and provide a basis to develop a design for a Smart Card application.

Determination of the particular requirements of the lease environment will tie this research into the ongoing developments of Smart Cards in Electronic Commerce.

Owing to the choice of an unproved developmental algorithm and associated processes a simulation of the design is necessary before beginning a prototype. The prototype will be the basis for developing an application which may integrate all the important characteristics of the design.

The goal is a Smart Card application to provide authentication, identification, secure communications and data logging in a distributed, mobile computing environment.

2.6. Scope of Proposed Research.

2.6.1. Introduction

The development of an integrated system to monitor and control access and use is limited by the requirements of the owner and user.

The main avenue for research is the 'Multi-user, Multi-Level, Multi-Resource' aspect of administering a loan or hire arrangement. A hire company may require that many people drive its vehicles under differing conditions. A cleaner must be able to access a vehicle but not drive it, a mechanic must be able to drive and service a vehicle and a customer must be able to operate the vehicle freely but not have complete control over the vehicle and its information.

Starting from the relationships governing the granting of access and use of a vehicle, investigation of the use of Smart Cards to provide access, data recording and integration in a real-world environment will provide information for further development.

The Owner - User relationship may be complex and the methods used to transfer all or part of the owner's privileges and administer the use of a mobile resource must be effective. This will rely on secure recording of any data necessary to support the administration of an agreement.

Existing protocols for authentication and identification deal with a substantial part of the work necessary and need only be evaluated in the context of the research. However, the granting of permissions from an owner to a user is within the scope of the research. The Owner must be able to grant, modify or revoke access and usage privileges as required.

Research will address using a Smart Card as an authenticating artefact, a physical token that permits access and usage and is able to communicate details to any authorised party. Additionally, the provision of a basis for further development of this project may be useful in other areas.

2.6.2. Smart Cards

Access control systems with security and flexibility to handle multiple users in a distributed environment have become necessary due to the wide range of options available to the owner of a number of vehicles. This environment is characterised by distributed, mobile assets with a wide range of owner and user expectations which include safety, economy, reliability and security.

Smart Cards provide computing power separate from the usual computer and monitor combination. Though the banking and finance industries provided initial development of Smart Cards and defined the common format, recent developments make Smart Card use possible in a wide variety of activities.

Access control and vehicle management based on a Smart Card may provide flexibility to the owner and user. A single card may be used to authorise use, record data, purchase needs and provide security for the Owner, User and vehicle. At present, the use of access control systems incorporates burglar alarms and electronic keys that broadcast the access code from outside a vehicle. A technique called 'sniffing' uses an electronic device to detect and store the access control signals that enable an unauthorised user to simulate an electronic key to the vehicle.

Current solutions to this problem include the use of 'Rolling Codes', a cryptographic protection of the access code where each iteration is different and a discernible pattern is difficult to detect.

Contact-less Smart Cards with reduced broadcast radius may provide a measure of security. Contact Smart Cards do not broadcast data and may provide greater security than conventional broadcast technologies but require that the card is placed into a card reading system before communication occurs. Reliability is a major consideration in both systems and contact cards and readers must withstand physical wear and tear.

2.6.3. Data Logging

Management of a complex and demanding distributed environment will rely on information from the operation of the resource for use in the resolution of a usage agreement.

This data includes monitoring of vehicle operating parameters as well as time and duration of operation, location of vehicle and name of driver. The availability of this range of data may complicate the administration of the agreement, for example, a dynamic charging structure where the actual usage determines the costing.

As an alternative to the 'unlimited kilometres' in some current agreements, the charge can be varied where the use occurs in town or country or at high or low speed. If the use occurs during excessive engine operating temperatures or low oil pressures then the user may incur a surcharge on the agreement.

Consequently, secure logging of vehicle operating parameters to record usage is important. Secure logging of data may include secure physical storage and the use of cryptographic or information storage techniques such as compression.

2.7. Summary

- ❑ The identification of the characteristics of the agreement and the needs of the parties to it is essential. Data must be verifiable at all stages of the usage period and securely stored.
- ❑ Smart Cards provide portable, secure computing power in a distributed environment with the ability to store and process data on or off-line.
- ❑ With ongoing broad spectrum development in automotive engineering, systems integration is becoming more common. Integration enhances efficiency and safety while providing opportunities for effective administration.
- ❑ Secure operation without imposing arbitrary limits on a user is needed and simulation must address the memory requirements and necessary data.
- ❑ Identification or development of a suitable model must address the requirements of Smart Cards and distributed computing.

3. Review of Literature

3.1. Introduction

The research topic is titled 'Development of a Model for Smart Card Based Access Control in Multi-User, Multi-Resource, Multi-Level Access Systems'.

Smart Cards are used to provide an intelligent token. 'In access control the card may be employed to open the door of a secure room or to obtain information from a secure database.' (Longley and Shain, 1987, p175)

Access control is defined as 'procedures to limit entry to ... or to limit use of a computer/communication system or computer stored data, to authorised personnel.' (Longley and Shain,1987, p2)

Multi-User indicates use or access by many different users consecutively or concurrently. Multi-Resource describes varied resources available to users with differing requirements. Multi-Level refers to different levels of permitted access or use of a resource which may depend on user status or other qualifiers such as time of day.

A typical application would be the hire or lease of vehicles and the associated management of information where identification, authentication and secure transactions are important.

Identification of all parties to an agreement is necessary to reduce the opportunities for fraud and ensure that any mandatory requirements are met. Identity is the 'condition of being a specified person'(TCOD,1977, p533)

Authentication must address continued identification of the user of a resource, the nature of the transactions and the data generated. Authentication is to 'establish the truth or authorship or validity or genuineness of [something]'. (TCOD,1977, p63)

A transaction is considered as a business process carried through to completion, for example, an agreement resulting in a lease / hire contract.

A transaction is '1. An event which results in a record being generated or updated in a data processing system. 2. The record so generated. 3. A set of exchanges between a terminal and a central processor.' (Chandor, 1981, p173) Additionally, a transaction is the 'Management of business' or 'piece of especially commercial business done.' (TCOD,1977, p1231)

Secure storage of information and access details is essential to the operation of the system. Secure means 'Safe against attack, ... , reliable, certain not to fail or give way' in a general sense (TCOD,1977, p1027) More specifically 'assuring the secrecy, integrity and availability of components of computing systems. The three principal pieces are ... hardware, software and data.' (Pfleeger,1989, p19)

The basic automotive system being examined is the 4-stroke engine with a proprietary electronic Engine Management Unit (EMU).

The conceptual supervisory controller will provide information to develop the basis of a design for an integrated Vehicle Management Unit (VMU) using Smart Card technology to administer system activities.

3.2. Derivation of Automotive Control Systems.

Initial development of Internal Combustion Engines came from the need for lighter, more efficient engines or power plants for mobile equipment and existing steam technology. (Strandh,1979, p139) A classification system of the various types of power plant shows the characteristics and relationships of engine types. (Appendix 9.2)

The development of automatic engine control systems has produced identifiable types of systems. Initial systems used basic mechanical devices with a human input to provide feedback and control. Among the earliest examples of automatic devices were mechanical steam pressure regulators to reduce the need for human input. (Strandh,1979, p184)

Improved performance required increased complexity of control systems. Purely mechanical systems gave way to hybrid electro-mechanical control systems for a more complex array of ancillaries. Examples of this include 'Magneto'- (Motor56,1956, p118) and conventional 'Contact Breaker'-based ignition(Motor63,1963, p51) in petrol engines, electrical lighting, fuel and performance monitoring equipment and accessories such as the windscreen wipers and hydraulic brakes.

Many of these accessories are electrical in nature and as early as 1921 specialised texts describing theory, operation and maintenance of these accessories existed (Motor56,1956). The increasing complexity of these systems provided opportunities for electronics to perform the control functions. Current-switching transistors increased the reliability and effectiveness of the ignition by increasing the current through the coil while reducing current through the points.(Motor63,1963, p61)

The increasing complexity of engines and associated electronics developed into the computerised control systems typified by the range of fuel injection products cited in Gregory. The development of Engine Management Units (Bosch93,1993, pp428-477) shows an increasing use of electronic technology and an increasing integration of vehicle systems to provide a safer, more economical vehicle. 'The development of the automotive diesel engine is governed primarily by requirements for clean exhaust, improved fuel economy, and the optimisation of driveability' (Bosch94,1994, p186) Compression-Ignition engines (diesels), which do not use spark ignition, may benefit from 'Electronic Diesel Control' (Bosch93,1993, pp501-505).

Substantial on-board computing power is coupled with numerous, complex sensing systems to provide control of vehicle characteristics. Security, safety and comfort systems include theft deterrent systems, safety systems (airbag, seat belt tighteners), navigation and communication systems, electrical locking and window control. The development and adoption of the Controller Area Network (CAN) standard for data communications throughout a vehicle emphasises the importance of digital control systems in automotive use. (Bosch93,1993, p776-778)

The trend is clearly toward greater integration of systems. Computerised control systems are expected to provide a whole Vehicle Management Unit (VMU). As future trends may not be readily discernible it would be expected that such systems be flexible, reliable and efficient to permit their widespread use.

3.3. Engine and Vehicle Management Units

3.3.1. Introduction

Engine Management Units (EMU) integrate the control functions of the disparate components of a vehicle power unit. The development of solid-state electronics has made it possible to provide rugged, reliable and versatile control mechanisms for the engine, transmission and ancillary systems of a vehicle.
It is expected that the EMU will acquire additional functions and become an integrated Vehicle Management Unit (VMU) which will provide a wide range of functions to the user on demand or by configuration.

3.3.2. Engine Types

Internal combustion engines can be divided into classes by cyclic or continuous combustion, auto-ignition or external ignition, diesel, hybrid or Otto-cycle type and heterogeneous or homogenous mixture type. (Appendix 9.2)

The main application is the cyclic combustion type which includes Diesel, hybrid and Otto-cycle engines which are commonly used in motor vehicles. The most common is the Otto-cycle (4-stroke), spark-ignition engine using petrol as a liquid fuel, followed by the two or four stroke compression-ignition engine using diesel fuel.

3.3.3. Ancillary Systems

Ancillary systems range from instrumentation providing information and services to complex interactive systems providing aid to the operator. These systems include cruise control, air-conditioning, power-steering, entertainment, sun-roof, window winding, central-locking, adjustments to seats and mirrors, adjustment to suspension or transmission, navigation information and others such as Closed Circuit TV for safety and monitoring of operating conditions.

The provision of these ancillary systems has increased the complexity of the control and reporting task. The workload in monitoring these additional systems can be lessened by integration of these systems into the vehicle control systems.

3.3.4. External Systems

A major external system is the management system responsible for the day-to-day accounting of the vehicle operations by the owner / user.

Other external systems such as traffic management systems and toll systems (Harrop,1994, pp31-32), while essentially stationary, require information interchange with the VMU. These systems may provide weather, road conditions and routing information, electronic toll charging, theft and illegal use monitoring.

For example, the SafetyCam (OD97,1997) system is in operation in New South Wales where imaging technology is used to record vehicle details at locations along the major highways. This information is used to monitor vehicle usage and traffic offences where a vehicle that passes a control point too soon is deemed to be speeding. Excessive hours on the road can be determined making log-book fraud difficult. Additionally, traffic offences such as using a mobile phone while driving a vehicle can be monitored.

External systems that communicate with a vehicle exist for electronic logging of vehicle movements and characteristics such as gross weight. Weigh bridges can be used to provide control of vehicle loads and consequently minimise damage to roads.

3.3.5. Trends in EMU development

The extra load placed on the driver to integrate the information from various sources has provided impetus to the development of integrated Vehicle Management systems that allow the user to select options and configurations.

The Cadec Systems 4000P has a proprietary 'Data Bridge' which 'is a communications gateway designed to cost-effectively link the engine, its subsystems, the driver and the vehicle through an onboard computer to the company's home office and its remote locations.' (Appendix 9.22)

The 'Safety Alert' system from Cobra Electronics Corporation is 'designed to improve highway safety by automatically transmitting warning signals to the radar detectors of motorists within 3/4 mile of the emergency or potentially hazardous site.' ' (Appendix 10.22)

An Onboard Electronic Scale marketed by Dynacraft provides accuracy of 1% and is used on vehicles with pneumatic suspension. The system can determine gross, payload and axle group weights ' (Appendix 9.22)

SCAN 'is an ultrasonic sensory device designed to assist drivers in safely changing lanes, backing up or docking... The module visibly and audibly alerts drivers to obstacles within SCAN's detection zone' ' (Appendix 9.22)

Satellite communications are catered for by Qualcomm's 'Enhanced Display Unit for the OmniTRACS satellite mobile communications system.. Used to send and receive text messages' it can be used with the on-board recorder and a smart card reader/writer to allow drivers to store or retrieve data. ' (Appendix 9.22)

Satellite navigation systems with onboard computing to provide guidance to the user and proposals have been made for stolen vehicle immobilisers which can be activated by a mobile phone link to the vehicle. (Jones,1998)

Personalisation of the vehicle to suit individual preferences may involve many semi-automatic systems such as steering wheel height, seat position etc. Additionally, off-the-shelf engine management units can be fitted to an existing system which permit dynamic alteration of the engine operating parameters which may be useful in optimising engine performance under changing conditions such as ambient air temperature. (Wolf3D, no date)

3.3.6. Summary

Increasing use of electronics has lead to computerised equipment and systems integration to reduce the workload on the driver and improve safety and efficiency.

Vehicle management systems routinely include interaction with ancillary and external systems. Integration of the systems is occurring and the necessary standards are being put in place.

- Programmability permits the user to select complex functions with simple actions, for example, cruise control of a vehicle. Modularity is where similar functions are grouped together for ease of maintenance and access.
- Provision of safety, security and comfort functions in addition to the engine and vehicle management functions is already well advanced. The practicality of such systems may depend on clearly defined standards to permit inter-operability and economic use of the systems.

3.4. Data Storage Requirements

3.4.1. Introduction

Administration of any system, hardware, software or paper-based, requires that all relevant data is stored in a suitable form to permit analysis and verification. Identification of the sources and types of data coupled with analysis of which data is necessary for record keeping will permit an estimate of the data storage required.

Additional functions or uses of the data may become apparent leading to the provision of efficient or novel facilities to the owner and user.

The need for secure transactions and data processing is evident in a distributed, multi-level, multi-resource environment. 'Other applications where authentication [of information] is important are alarm systems, satellite control systems, distributed control systems and systems for access control.' (PRENA,1991, p2)

3.4.2. Engine and VMU

The problems determining data storage requirements relate to the requirements of the EMU, the ancillary and external systems. The most important consideration is the amount of information required by external systems such as management systems and depends on the amount of data required and the rate at which it is stored.

Data-transfer rate is defined as the number of bits per second where a bit is a 'Unit of information expressed as a choice between two possibilities' (TCOD,1977, p99) and 'One of the two characters (0 and 1) used in binary notation. Also signifies the smallest unit of data' (Chandor,1981, p25) A byte is taken to mean eight bits.

A sensor with 8 bit resolution can provide 256 different values. Reading this sensor every second provides 3600 bytes per hour. Ten sensors would provide 36000 bytes of data for every hour of operation. During long-distance use of a vehicle with multiple shifts over a week the data would be approximately 5.77 Megabytes of information where a Megabyte is 1048576 bytes.

Additional data such as driver details, accommodation, meals, date and time of operation, location, load and fuel use may substantially increase this value.

The number of sensors used with a VMU would be vastly increased with the integration of ancillary and external systems sensors. Legal requirements would also place constraints on which data is stored for audit, mandatory business and taxation records and resolution of over-weight, over-speed and over-time complaints.

This information cannot be stored in the limited memory of existing smart cards and some additional storage is necessary. Existing Smart Cards may have as much as 8 kilobytes of EEROM and 20 kilobytes of ROM. (Hendry,1997, p106) To deal with the data storage requirements data compression, large capacity memory storage and remote reporting are considered.

3.4.3. Data Compression

To conserve storage space and provide some coding of information various data compression algorithms can be applied. 'Data compression refers to the storage of data in as little space as possible. ' (Lew,1985, p300)

Run-length encoding of data converts a string of symbols such as 'AAABBBAABBBBB' to '3A3B2A5B' where each consecutive occurrence of the symbol more than twice is encoded as the count and the symbol. (Sedgwick,1990, p320) This is a simple compression method and is suited to periodic recording of data that does not change quickly. Periodic temperature monitoring can record duration and change of temperature by the number and value of the symbols.

More complex methods include 'Huffman Codes' which store data based on the frequency of occurrence of each character where the most common character is stored in the least bits. (Lew,1985, p355; Kruse,1987, p453) Different series of data may be encoded by different Huffman codes and unless the specific Huffmann coding is recorded and used to decode the sequence the data may be unintelligible (Sedgewick,1990, pp325-330).

Additionally, an erroneous bit in a Huffman coded sequence may cause error propagation and substantial delay may occur when the data has to be uncompressed or corrected for evaluation.

3.4.4. Remote Reporting

Remote Reporting could be achieved with a dedicated transmitter / receiver providing periodic updates of the data to a home base during operation. (Qualcomm Appendix 9-22) An alternative could be a mobile phone dedicated to vehicle use which could be periodically used by a VMU to report to home base or even provide 'panic button' or accident occurrence notification to emergency services.

The GSM - Global System for Mobile Telephony is an important standard for mobile telephone systems. The individual phones are identified with a Smart Card which functions as the Subscriber Identity Module (SIM) (Hendry,1997, pp150-152)

External systems such as road toll collection may need to identify the vehicle and driver to determine a payment method and a traffic monitoring system may impose a speeding fine and notify the user with a message or tone inside the vehicle. A consideration with the automatic assessing of charges is whether the User will dispute the alleged event and use the system data for arbitration.

3.4.5. Information available from EMU

Information available from the EMU is principally viewed as information derived from sensor data after processing. Normally, the information is used to alter the engine characteristics for efficient running under the current operating conditions and to update information on the various displays.

The sensors for the EMU provide coolant temperature, oil pressure and temperature, crankshaft position, engine rotation and rate (Revolutions per Minute - RPM), exhaust gas oxygen content, fuel mass, air mass and temperature, throttle position and transmission range selection. (Gregory,1990; Bosch93,1993)

The types of data are:

Raw Data	-	data from the sensors before processing (Chandor, 1981, p150)
Sensor Data	-	data from the sensors after processing
EMU Data	-	data generated by the EMU
Derived Data	-	data inferred from other types of data

Raw Data is the actual output of a sensor. The sensor output may be inadvertently modified by the characteristics of the sensor system and care must be taken to ensure that monitoring this data does not alter the quality of the data provided to the EMU.

Sensor Data is the data up to the input of the EMU processor and may include on-board signal processing and conversion from analog to digital forms. The effects of monitoring such data must be carefully considered as unintentional changes may affect the smooth running of the vehicle.

EMU Data includes the output signals to the control and reporting devices, that is, fuel injector signals and information to the display system for the driver.

Derived Data is all data that can be inferred from the previous data. It may be deduced from engine rotation of 5000 RPM and a road speed of 0 Kilometres per hour (kph) that the engine is being 'revved' in an unloaded state and this may be considered an abuse of the vehicle.

It is noted that all of these information types may be used to provide diagnostic and maintenance support.

3.4.6. Information available from ancillary systems

Automated systems may use Sensor, EMU and Derived data to alter the settings to account for temporary changes in the operation such as air conditioning, electrical load (heater, blower, headlamps), power steering load, engine temperature, transmission and clutch status. (Gregory,1990, p24)

Air-conditioning system information includes whether the compressor is activated or the fan is being used. The loading effects of compressor operation may affect engine idle speed depending on the amount of cooling or heating required. Variations in idle speed of the engine are sensed and controlled by the EMU. (Bosch93,1993, p466)

For example, cooling the vehicle while the window is open could induce a predetermined level of cooling until the window is shut. Derived data could be generated from air-conditioning system, central locking and power window controller data.

Information from the braking systems may be brake application, intensity and duration which could provide important insight into vehicle operations. Air pressure values for pneumatic brakes are important to the driver and brakes failing to release correctly may cause inefficient operating conditions or possibly an accident. (Bosch93,1993, p604)

Suspension systems provide data about the current load, usage and state of the vehicle. Benefits include reduced energy consumption during braking and acceleration, reduced fuel consumption by reducing vehicle height with increased speed and enhanced stability during cornering. (Bosch93,1993,p562)

Air bag suspension on heavy vehicles may provide data about vehicle load distribution across each axle and whether the suspension becomes unsafe or fails.(Dynacraft Appendix 9.22)

Transmission systems provide information about the current range selected and road speed which is used to adjust engine operating parameters. Automatic transmission provides 'kick-down' and efficient running modes. Kick-down is the automatic selection of a lower transmission range to maximise acceleration during overtaking. Efficient running includes transmission modes where for long-term high-speed driving the lowest range is rendered inoperative to reduce the load on the engine.

Central locking systems may indicate whether the doors are correctly closed as well as locked which may affect the performance of the vehicle under certain operating conditions. (Bosch93, 1993, p725) For example, Transperth buses operating in Perth cannot be driven with the rear passenger doo: open as the brakes are automatically applied until the door is closed.

3.4.7. Information available from external systems

At present, external systems are not plentiful, though the range of systems is increasing. For example, satellite navigation systems providing directional guidance to emergency vehicle operators can be used to relay the position of the vehicle to a control system.

Identification of vehicles and loads may be automatically registered as the vehicle passes a control point. Additionally, some information systems may not interact with the vehicle, such as traffic monitoring systems that determine speed directly (radar or laser) or indirectly by video surveillance systems like SafetyCam. External systems represent a growing aspect of the opportunities provided by technology to optimise the operation of motor vehicles.

3.4.8. Operational and Environmental Considerations

A complete specification of memory chip development would have to take into account all the developments to date. However, a table provides an overview of the principal semiconductor memory device types. (Appendix 9.3)

The principal considerations here are the ability of the chip to survive in hostile environments where temperatures range from -40C for operation in an arctic environment such as winter in northern Canada to + 130C (Bosch93,1993, p114) where fluids from an overheated engine have leaked out. Environmental testing is already a standardised procedure. The 'DIN IEC 68 - Environmental testing procedures for electronic components and equipment' is an industry standard. (Bosch93, 1993, p350)

Exposure of system components to vibration, dirt, petrochemicals, hot liquids, accelerations and tampering / abuse from the user mean that the components need to be designed and selected for reliability, durability and maintainability. Additionally, if information is used for the resolution of usage agreements, then the storage and data must resist degradation, intentional or unintentional and from natural or unnatural causes.

With the diverse electrical environment being subject to substantial changes, the design and selection of the chips must allow for wide ranges of supply voltage and current variation as loads are switched in or out. Additionally, very high voltages up to 30 kilovolts are available from the ignition components (Bosch93, 1993, p447) and very high frequencies from communications devices such as mobile telephones and radios.

In Australia, Citizen Band (CB) radios transmit from 26.965 MegaHertz (MHz) to 27.405 MHz in the High Frequency (HF) band and 476.425 MHz to 477.400 MHz in the Ultra High Frequency (UHF) band (Smith,1996, p238). Mobile phones and GPS navigation systems the UHF Band which covers from 300 MHz to 3 GigaHertz (GHz) (Young,1979, p187).

To conclude, memory for use in the hostile and rapidly changing automotive environment must be physically robust. The amount and type of memory needs to be carefully considered in the light of ancillary and external systems and legal requirements.

3.4.9. Sensor and Interface Developments

A sensor or 'Transducer' is part of the measurement system containing 'three major elements: an input device, a signal conditioning or processing device and an output device.' (Cooper and Helfrick,1985, p348)

The sensor or transducer is a 'Device to convert variations in one quantity into those of another' (TCOD,1977, p1231) and more specifically 'Any device that converts a non electrical parameter, e.g. sound, pressure, or light, into electrical signals or vice versa.' (Young,1979, p540)

Interfacing of the varied sensor signals to the EMU occurs throughout the system and different requirements exist. The range and capability of sensor systems is improving and 'smart' systems use sensors with built-in processing power coupled to a serial bus Controller Area Network (CAN) (Bosch93,1993, p776). Sensors with built-in microprocessors providing independent signal processing are clearly described. (BOSCH93,1993, p 103)

3.4.10. Data Storage Summary

- ❑ Sensor development and integration is required because of the amount of information needed to dynamically alter the operation of the vehicle to meet changing conditions.
- ❑ Substantial broad spectrum development is already taking place.
- ❑ Integration of systems is desired for safety, economy and ease of operation
- ❑ Suitable standards and protocols for increasing the range and ability of computers have been considered in the development of the range of sensors which incorporate facilities such as error detection and handling, fault confinement and fault tolerance.
- ❑ Memory chips must be selected to meet stringent environmental requirements and implementation of the design will require a high level of engineering skill.

3.5. Agreements and Contracts

3.5.1. Introduction

A vehicle owner may permit the use of the vehicle without supervision in a hire/lease agreement which is a 'contract legally binding on parties' (TCOD,1977, p21). The contract consists of 'an agreement which gives rise to legal rights and obligations between parties to it and which will be enforced by the courts.' (Gibson & Fraser, 1995, p826)

A contract transfers some of the Owner's rights over the vehicle to the User and must address the potential usage. More information than the distance travelled or the fuel used may be needed.

3.5.2. Explanation of Terms

3.5.2.1. Possession and Ownership

- ❑ Possession in law primarily means physical control over a resource.
- ❑ Ownership of a resource means the rights to exclusive enjoyment of the property. (Gibson & Fraser,1995, p642)

3.5.2.2. Identification and Authentication

- ❑ Identification means 'Every subject must be uniquely and convincingly identified. Identification is necessary so that subject/object access request[s] can be checked.' (Pfleeger,1989, p283)

- ☐ Identification is based on something :
- ❖ Known -PIN number, password or some secret knowledge
- ❖ Possessed - artefact containing secret knowledge such as a Smart Card or implying identity such as a uniform or identification document
- ❖ Inherent - physical characteristics such as finger prints
- ☐ Identification protocols must be reciprocal, that is, able to be used to identify all parties to the agreement by any one party. Computational efficiency is desirable. (Menezes, Van Oorschot & Vanstone, 1997, pp387-388)
- ☐ Authentication relates to entities and information. 'Entity authentication is the process whereby one party is assured (through acquisition of corroborative evidence) of the identity of a second party involved in a protocol, and that the second has actually participated (i.e., is active at, or immediately prior to, the time the evidence is acquired.)' (Menezes et al., 1997, p386)
- ☐ 'Data origin authentication is a type of authentication whereby a party is corroborated as the (original) source of specified data created at some (typically unspecified) time in the past.' (Menezes et al.,1997, p361)

3.5.2.3. Signatures and Authorisation

A signature links information to a particular entity and may be used to signify assent or acceptance of an obligation. (Appendix 9.4)

Mitchell, Piper and Wild suggest that 'The crucial properties of a written signature are that it is easy to produce, easy to recognise but difficult to forge' (IEEE92, 1992, p330)

Digital signatures must be easy to compute, easy to verify and have an appropriate lifespan. (Menezes et al., 1997, p30) A digital signature must also meet the requirements of a written signature.

Authorisation is the conveyance of permission to another person to do or be something.

(Appendix 9.4) Authorisation to use a resource must be clearly traced from the owner to the person with the right to use or possess it.

3.5.2.4. Non-Repudiation

Non-repudiation means that neither party to an agreement can deny accountability for an agreement. (Appendix 9.4) This means that an agreement cannot be altered except with the consent of the parties concerned and the agreement with any changes can be verified by an independent party. Consequently, recording of all information applicable to the agreement including changes and identities is important. The information must be stored in a manner that resists alteration yet is still readily accessible.

3.5.3. Electronic Commerce Considerations

In the past twenty years banks and other financial institutions have come to rely on electronic methods of transferring funds. These methods, while efficient, must engender confidence in the system by the users.

Computer crime is 'any illegal, unethical or unauthorised behaviour relating to the automatic processing and/or the transmission of data'. (De Schutter,1991, p3)

Electronic funds transfer has many benefits but problems fall into four categories: 'issues of a legal and contractual nature, consumer information problems, general economic problems and social problems.' (OECD, 1989, p13)

Major concerns exist about the nature of the contract or agreement between the issuer and user of the card as unilateral variations to contracts are possible. Information gathered to administer a contract may be used for purposes which may disadvantage the parties to the contract. The continued possession and use of the information relating to the agreement raises questions about privacy and security. Recommendations addressing these issues have been formulated. (OECD,1989, pp130-136)

Use of encryption technologies is subject to world wide scrutiny owing to the possibility of crime remaining undetected. Major concerns exist in the use of encryption as an aid to confidentiality of personal data contrasted with the legitimate concerns of the services tasked with maintenance of law and order. (COA,1996, p3)

Increased availability of electronic data processing equipment coupled with development of strong cryptographic methods has provided data security for the large amounts of data generated. The development of encryption methods may be summarised as 'stronger, faster, cheaper' (COA,1996, p27)

Smart Cards have uses in identification and authentication of information and transactions. Uses as part of a cryptographic system and in anonymous transactions are considered. (Chaum,1991)

3.5.4. Summary

Contracts and agreements which rely on the law for validation must be meaningful in a legal sense.

□ Information used to administer the agreement must be suitable and verifiable.

□

□ Electronic commerce must meet stringent requirements relating to the gathering and use of information.

3.6. Smart Cards and their Capabilities

3.6.1. Introduction

Rankl & Effing describe the first patent for an integrated circuit identification card in 1968 by Dethloff and Grötrupp (Rankl & Effing, 1997, p3), which preceded the similar application by Arimura in Japan in 1970. (Zoreda & Oton,1994, p 36) The Smart Card concept, a portable computer with a reduced set of capabilities and interfaces, has since overcome the technical problems and matured into a practical technology.

'Smart Cards may have two forms; one for a set of banking operations and the other, termed an intelligent token, can provide access control, perform encryption and authentication operations, etc. A multipurpose Smart Card for banking and financial transactions might perform one or more of the operations: (a) electronic chequebook; (b) electronic wallet; (c) electronic purse; (d) electronic token; (e) telepayment; (f) internal transfer; (g) access; (h) remote banking; and (i) portable file.'
(Longley and Shain, 1987, p317)

According to Guillou, Ugon and Quisquater, the Smart Card can perform only five operations:

1. 1. Input data

2. 2. Output data

3. 3. Read Data from Non-Volatile memory (NVM)

4. 4. Write or Erase in NVM

5. 5. Compute a cryptographic function
(IEEE92, 1992, p565)

Using Smart Cards for vehicle control is shown by products such as the Driver Card Manager marketed by VDO Kienzle (OD97,1997) and the ATMEL(EA,1997, p30) system for keeping data about vehicles on Smart Cards. 'Many of these applications are demanding more and more 'intelligence' in the embedded micro...'(EA,1997, p30) and applications such as road tolling and parking are readily available.(Hendry,1997, p205)

The limits imposed on the card by the component technologies have hampered the use of the card. For example, the average memory capacity of a Smart Card is measured in kilobytes and the interface is serial in nature reducing the bandwidth of the data transfer signals. (Zoreda & Oton,1994, p12) A common clock frequency of 3.5712 MegaHertz results in a maximum rate of 446400 bits per second.(Rankl & Effing,1997, p411)

3.6.2. Types of Cards

3.6.2.1. magnetic stripe cards

Data is encoded on a magnetic stripe on the rear of the card. The data is formatted into tracks which are read when the card is 'swiped' through a card reader and information can be written to or read from the card. (Hendry,1997, pp35-40)

3.6.2.2. optical cards

A 'Write-Once, Read-Many' storage device that permits a large amount of data to be stored. Operations are performed by a low power laser which reads the data by detecting the irregularities in the optical surface and writes by burning additional irregularities into the surface. (Hendry,1997, p 41)

3.6.2.3. Chip cards - memory only

Memory-Only Cards are analogous to a floppy disk in that any data stored on them can only be manipulated by an external system attached to the card reader. As a portable data storage device they can be used to transfer and store data. For example, patient records in a hospital where large amounts of data for each patient are recorded. (Zoreda & Oton,1994, p5)

3.6.2.4. Chip Cards - with microprocessor

Chip Card with a programmable microprocessor for managing and access control of data.(Zoreda & Oton,1994, p 233; Rankl & Effing,1997, p13) Processor cards are analogous to a small computer excepting that there is no internal power supply or display. Memory includes Read-Only Memory (ROM) which is permanent, Erasable Programmable ROM which is for long-term storage that may change only under special conditions and Random-Access Memory (RAM) for short term storage of working data.

3.6.2.5. PCMCIA Cards

The Personal Computer Memory Card Industry Association specifies standards which allow small memory cards and peripheral devices to be incorporated into a card format. Any type of semiconductor memory may be used and the storage capacity may be in MegaBytes. PCMCIA cards are used to transfer data between computers. (Hendry,1997, p46)

3.6.3. Standards

The International Standards Organisation (ISO-OSI) publishes standards relating to the design and use of Smart Cards such as :

ISO 7810	Identification Cards - physical characteristics
ISO 7811	Identification Cards - recording technique (magnetic stripe cards)

ISO 7816	Identification Cards - integrated circuit cards with contacts
ISO 9564	Banking - PIN management and security
ISO 10202	Integrated Circuit Cards - Security Architecture for financial transactions
ISO 10536	Identification Cards - contact-less integrated circuit cards

The Centre Européen pour la Normalisation (CEN) publish standards such as :

EN726	Identification Card Systems - multi-function cards
EN1038	Identification Card Systems - payphones
EN1546	Identification Card Systems - Intersector electronic purse.

3.6.4. Uses

Smart Cards may be used in many areas : identification and electronic cash for electronic commerce (Chaum,1991) Password authentication (Chang, Hwang and Buehrer,1993) Personal Communications security (Cooke and Brewster,1994) Multi-function cards (Cordonnier,1991) and Cryptography (Vedder,1992; Konigs,1991) among others.

Use of a Smart Card in the development of the proposal may require all these activities. Additional uses may be determined but the principal area of development is the multi-level, multi-resource, multi-access environment for vehicle hire or lease.

3.6.5. Summary

Consequently, the basis for the Smart Card based identification, authentication, authorisation and recording system is defined. Smart Cards will function as a key for user access to any permitted function while recording the actions and providing information to administer a usage contract. Additionally, Smart Cards will provide secure access to stored data and a basis for the development of an integrated VMU.

3.7. Existing Protocols for Information Transfer

3.7.1. Introduction

The development of the proposed research relies on using either existing protocols or developing a new ones to ensure that the information used is handled correctly. Fraud and other unwanted outcomes may result from a poorly implemented protocol.

To counter this problem it is essential to use existing protocols which have been exposed to critical analysis and appear to have well-described characteristics. Due to the limited time available existing protocols will be used wherever possible as development and validation of a new protocol would require substantial resources.

Consequently, only the relationships between the User, Smart Card and Resource will be discussed.

3.7.2. Existing Protocols and Algorithms

This list identifies some of the existing methods. Identification of suitable protocols is needed.

3.7.2.1. Identification

□ Feige-Fiat-Shamir

□ Fiat-Shamir

□ Guillou-Quisquater

- ❑ Schnorr (Schneier,1996)

3.7.2.2. Signatures

- ❑ RSA
- ❑ Rabin
- ❑ Merkle (Menezes et al., 1997)

3.7.2.3. Electronic Funds Transfer

- ❑ Secure Electronic Transfer (Whyte,1997)

3.7.2.4. Encryption

- ❑ Rivest-Shamir-Adelman (RSA)
- ❑ Data Encryption Standard (DES)
- ❑ Public Key
- ❑ Stream Cipher

3.7.2.5. Hashing

- ❑ MD4
- ❑ MD5
- ❑ Snefru
- ❑ RIPE-MD (Schneier,1996)

3.7.3. Summary

As far as practical, existing protocols will be used to minimise the amount of developmental work needed.

3.8. Cryptography and Data Security

3.8.1. Introduction

Cryptography is a generally available and applicable method of securing data and granting or restricting access to a system. Confidentiality, Data Integrity, Authentication and Non-repudiation are the goals of cryptography. (Menezes et al.,1997, p4)

Cryptographic protection must take into account the likely threat to the 'Confidentiality, Integrity and Availability' (PRENB,1991, p 6) of the data. A 'Cost-Benefit' analysis should be made to determine the costs of providing protection and the risks and costs of any losses. (Pfleeger,1989, p 466)

The requirements of secure transactions in an electronic and computing environment are important and many texts exist addressing the requirements. (Schneier,1996; Seberry and Pieprzyk,1989; Denning,1982)

The design requires the use of Smart Cards to provide the User with services to permit the operation of vehicles in a 'multi-user, multi-level, multi-resource' environment. This environment must include accurate and reliable information about the hire / lease agreement and the actual use of the vehicle for the resolution of the agreement.

Any assessment of cryptographic functions requires an understanding of randomness and the means of quantifying it. (Knuth, 1973, pp1-170; Lew,1985, pp266-267) This is a complex field and will only be addressed in a manner commensurate with the complexity of the design. Any assessment of the efficiency or practicality of the proposed system must address the operation of the system and cryptographic protection provided.

3.8.2. Hardware

Hardware systems to provide cryptologic activities are split between dedicated hardware or computers and microprocessors that can be configured to perform cryptographic functions. The hardware systems range from small add-on items which apply a form of stream cipher on a data stream to dedicated, purpose-built electro-mechanical and electronic machines.

Configuration of a general PC to perform cryptographic functions requires suitable software though this may include specific add-on cards or systems. Hardware based Data Encryption Standard (DES) systems (Kahn,1996, p981) are available and one of the main reasons for dedicated hardware is higher processing speed. Schneier cites a Digital Equipment Corporation prototype chip processing 1 gigabit per second compared with 'a software implementation of DES on an IBM 3090 mainframe can perform 32,000 DES encryptions per second' where each block is 64 bits and this equates to approximately 2.05 megabits per second. (Schneier,1996, p278).

3.8.3. Smart Cards

Applications of Smart Cards to commercial enterprises such as banking employ a cryptographic environment readily implementable on the card itself.(Rankl & Effing,1997, p66) Smart Card Standards include ISO 8372 describing cryptographic modes of operation, ISO 10202 describing security architecture for chip cards and ANSI 3.92 Data Encryption Algorithm (DEA) (Menezes et al.,1997, pp645 - 656) This applies cryptographic security to transactions that can be performed off-line and remotely.

Smart Cards exist incorporating arithmetic co-processors and algorithms for solving common cryptographic problems coupled with a full range of security devices. (Hendry,1997, p84)

3.8.4. Application Specific Integrated Circuits (ASICs)

Application Specific Integrated Circuits are purpose-built chips to provide specific complex functions. Cryptographic applications, which are by nature complex transformations of data, are available as ASICs.

The United States has developed two chips to implement a national secure communications standard. The 'Clipper' chip is an NSA-designed, tamper-resistant VLSI-chip designed for encrypting voice conversations; ' (Schneier,1996, p591)

The 'Capstone' chip includes the Skipjack algorithm in four modes, a public-key Key Exchange algorithm, the digital signature algorithm, the Secure Hash Algorithm a general purpose exponentiation algorithm and a general purpose, random-number generator that uses a pure noise source. (Schneier,1996, p594)

Additionally, ASICs provide many implementations of special purpose controllers. (EA1997, p101)

3.8.5. Software

Software for cryptographic applications can range from the simple, 'home-made' variety to the complex systems employed by government and big business. In the last 20 years public debate and awareness of cryptography and the necessary techniques has increased. It has ceased to be the province of the military and government agencies and become a wide-spread adjunct to the use of modern communications devices. (Schneier,1996; Denning, 1982)

Software exists to provide encryption & decryption, hashing, identification and authentication and other methods to carry out secure electronic transactions. A selection of cryptographic software is readily available (Schneier,1996, pp623-673) and may be purchased in floppy disk format quite cheaply.

Schneier provides 30 symmetric algorithms, 3 public key algorithms, 8 one-way hash functions and 4 complete cryptographic systems as well as various software and references. The references include 'Defence Trade Regulations', the 'DoD Orange Book' and the 'European Computer Security Green Book'. (Schneier,1996, p760)

The legal position of these methods depends wholly on the use to which they are put and the country in which the activity takes place. The Australian position is set out in the 'Review of Policy relating to encryption technologies' published by the Commonwealth of Australia Attorney General's Department. (COA,1996)

3.8.6. Algorithms

3.8.6.1. Introduction

Algorithms need to be selected to provide for secure data storage, data transfer and other functions as needed. Hashing and encryption algorithms need to be assessed in the context of the proposed method.

3.8.6.2. Secure Hashing

The hashing algorithm, MD5, is a readily available method and the availability of the source code complete with test data means that it can be tested in any implementation. (RFC1321,1992) Rankl & Effing suggest that the minimum needed for an Assembly language version of a hash function would be 4 kilobytes.(Rankl & Effing,1997, p84)

Additionally, usable MD5 software can be down-loaded from the Internet. (Gnomeworks,1998)

3.8.6.3. Electronic Funds Transfer

The purpose of Secure Electronic Transfer (SET) is to provide confidentiality of information, ensure payment integrity and authenticate both merchants and cardholders. (SET96,1996 p3) SET uses the RSA algorithm.

A computationally more efficient implementation called 'LITESET' is being developed.
(Hanaoka, Zheng & Imai, 1998)

3.8.6.4. Lease Agreement

This is to be developed to implement the proposed simulation project. It is a simple exchange of authentication, identification and transaction details coupled with an electronic signature or validation.

3.8.7. Summary

- ❑ Cryptography is a readily available and cost effective method of securing information.
- ❑ Cryptographic functions can be implemented on Smart Cards. Smart Cards can be used as a component of a cryptographic system.
- ❑ Legal restrictions may apply on the use of cryptography and associated materials.
- ❑ Verification of algorithms is essential to the ongoing development of the developmental algorithm

3.9. Summary of Literature Search

- ❑ Substantial broad spectrum development is already taking place.
- ❑ The trend is toward greater safety and efficiency of vehicle use by increasing the number and capabilities of the microcomputers in a vehicle.
- ❑ Safety and security are of great importance. Safety in operations and security in the data transfer and administration of the vehicle requirements.
- ❑ Integration of the desired and necessary systems is proceeding but appears to be piecemeal. A clear integration and development regime is needed and currently standards are being put in place to ensure inter-operability
- ❑ Multiple function Smart Cards are being developed.
- ❑ Any algorithm used in the proposed environment will need to be verified before use. Verification may lead to certification.
- ❑ A single card implementing all the necessary functions to control a vehicle in a distributed environment is not currently available.

3.10. Research Questions

3.10.1. Introduction

As a result of the literature search it is apparent that there is a need for an integration of some of the themes inherent in Smart Card based access to vehicles. Broadly speaking, the opportunity for a secure, multi-function card to control all access is evident. These questions are quite broad in their scope and the subsidiary issues and conclusions are covered in Section 7.3 of the thesis.

3.10.2. Questions

1. What are the characteristics of a M Users, N Resources, P level access environment and which authentication model, if any, could effectively represent this relationship in a real-time, distributed computing environment?
2. Investigation into the requirements of authentication and digital signatures to determine the technical considerations that may have a bearing on the security of the proposed model?
3. What requirements are necessary for untamperable electronic data storage for short and long term storage to assist in resolution of disputes in the transaction cycle?
4. What data from the operation of a prototype is of importance in effecting the proposal and which security issues need to be addressed?

4. Overview of System Design

4.1. Introduction

Developing a practical design requires an understanding of the Smart Card development process coupled with electronic, automotive and software engineering, systems design and analysis of the information transactions. The most immediate limitations are memory storage requirements and the rate at which information can be stored and transferred through the system.

An estimate of memory storage requirements depends on the actual hardware configuration. Listing all the sources of information that a system might use and the information transfer rates can be used to estimate the system characteristics.

Transactions are any operations involving the transfer of data within the design and characterise the uses of the system.

4.2. Initial Hardware Design Of Complete System

In figure 4-1, the Engine Management Unit (EMU) uses microcomputer technology to control the engine and transmission. The EMU processes information from the sensors on the engine and transmission to determine the necessary fuel quantities and ignition timing to achieve efficient use while responding to changing user needs.

The proposed Supervisor Unit (SU) is required to supervise the EMU and process the additional information necessary to administer the use of the vehicle. It is expected that the roles of the EMU and SU will merge and all activities affecting vehicle usage will be controlled by the Vehicle Management Unit (VMU).

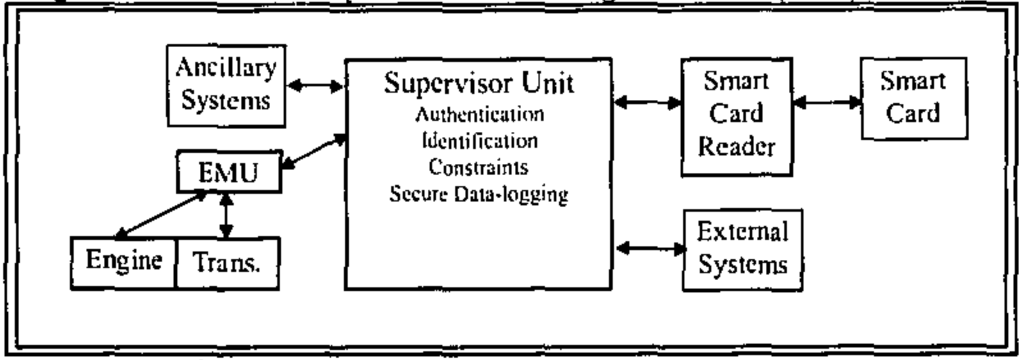


Figure 4-1 Schematic of Complete System

A secure environment is needed to store user identification and associated constraints, vehicle usage and authentication of transactions to assist in administering the usage agreement.

Ancillary systems are part of the vehicle controlled by the VMU and include comfort, safety, security and convenience systems. External systems which interact with the VMU may include traffic management systems, toll collecting systems, satellite navigation and emergency vehicle systems.

The detailed design of the VMU will not be addressed within this project as the design of a Smart Card system addresses many of the main requirements which include secure access to the VMU and its information.

Smart Cards for use in the design may have as little as 2 kilobytes and as much as 20 kilobytes of data storage. (Hendry,1997, p106) The data transfer rate with a Smart Card is limited by the implementation and this can be as low as 9600 Baud, where one Baud is equal to one data bit per second. A fundamental clock frequency of up to 5 Megahertz is expected but this is modified by the Elementary Time Unit and other time units defined in the applicable standard. (Zoreda & Oton,1994, p 78)

A complex VMU will have multiple data flows and the data transfer rates are limited by component technologies and external implementations.

Memory on the VMU can store substantial amounts of data but the data must be communicated to some user system. An alternative to the slow rate of transfer in the Smart Card is to use the greater transfer rate of a mobile phone and modem to communicate between the VMU and a remote base station. The use of the Mobile phone and Modem will not be addressed in this design.

4.2.1. Memory Storage

Alternatives to the use of custom made components to implement the proposed system include a Motorola 68hc11 microcontroller with up to 1Mbyte of external flash memory which has excellent power-off memory retention. (Hands,1997) Articles in commercially available magazines also address the issue of adding memory to micro-controllers.(Sibigtroth, 1997)

4.2.2. Prototype Hardware Basis

A prototype system can be developed on a Personal Computer (PC). A card-sized PC provides a 50 megahertz 486 processor and all the facilities to perform the on-board supervisory functions and administer the ancillary and external systems. This option allows software to be written in a high level language and permits rapid development of the system on standard computers and compilers. (OKIDATA, 1997)

The initial prototype configuration would be a PC without peripherals such as a hard disk drive, floppy disk drive or even a monitor. The initialisation of the system will be from the onboard Basic Input Output System (BIOS) and a program stored in non-volatile memory. The data storage requirements would be a non-volatile, re-writable memory component such as flash memory.

Alternatively, solid-state disc drives are available to permit almost normal operation of a PC based prototype for the VMU (Circuit Cellar ,1997, p36).

The data handling can be simplified by using a serial data bus which will reduce the amount of wiring necessary to install the system. Flash memory with serial Input/Output (I/O) is available (Cantrell, 1997) from Nexcom and offers 4 or 8 Megabit Chips with substantial write protection to prevent data errors.

4.2.3. Prototype Software Basis

A software simulation of the design will help determine the necessary data to be processed to meet design objectives. The principal component of the system is the Smart Card which is expected to provide a User with secure access to the system. Additionally, the Smart Card may be used to provide additional functions relating to the operation of the Vehicle in a distributed, Multi-user, Multi-resource, Multi-level environment.

4.3. Analysis of a Transaction

4.3.1. Limitations

Identification of the parties to an agreement is outside the scope of the research. It is, however, covered by existing protocols. The most common method used to identify a person requires documentary evidence such as a Driver's Licence with some form of picture and address details, a birth certificate or official document such as a passport.

Transfer of funds in a distributed environment is covered by existing protocols such as Secure Electronic Transactions (Whyte, 1997).

Consequently, identification of the Smart Card and storage of its details together with the transactions necessary will be covered.

The Entity Relationship diagrams describing the relationships are included in the Appendix 10.6.

4.3.2. Description of a Possible Agreement Structure

Generally, Person A and Person B negotiate some type of agreement which may be informal or formal. An informal agreement may be a short-term loan and a formal agreement may need a written contract to make it valid or enforceable.

Any agreement may need to be evaluated to ensure that it is complete, binding and meaningful. The agreement may need to be authenticated to a third party. Authentication must ensure that the agreement record is not changed except by mutual consent of all the parties and the changes cannot be concealed from an auditor. The assumption is that any or all of the parties to the agreement may not be honest and an observer may not be well-intentioned or accurate.

In figure 4.2, Person A and Person B intend to agree about the use of a resource in the custody of Person A and an agreement listing the permitted options and obligations is drawn up.

Should B attempt to evade responsibility for use of the resource by claiming that A has unilaterally altered the contract then A relies on a copy of the contract signed by B, and an independent witness, C, who is acceptable to both A and B.

Authentication of the agreement between A and B involves examining the signatures of both parties and the credentials and identity of C who is expected to give an unbiased report on the validity of the agreement and signatures.

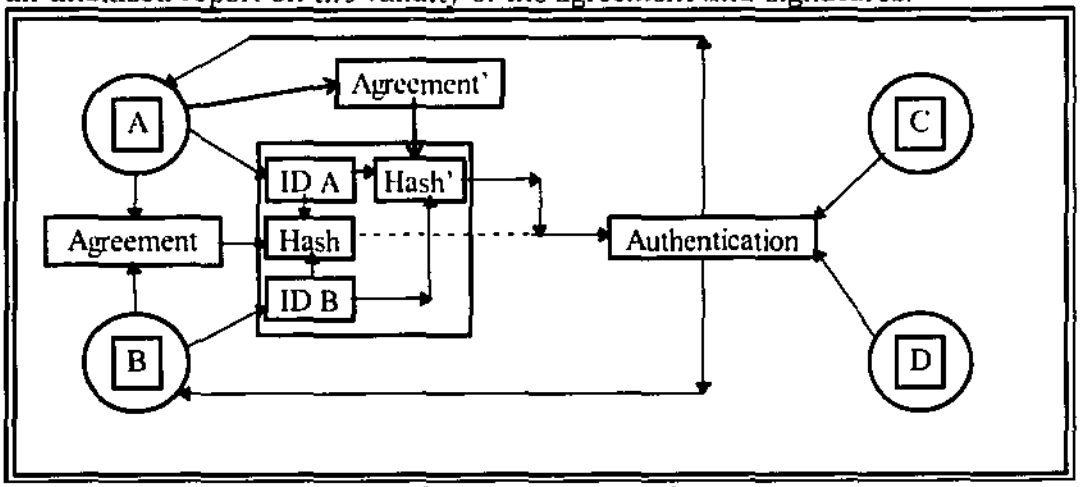


Figure 4-2 Agreement and False Agreement Schematic

Should A or B conspire with C to act against the other party, C relies on an unblemished reputation to support the acceptance of an 'unbiased' opinion. The malpractice is effected by claiming to D, an arbiter, that if C is reliable then so is the person supported by C. Person D is defined as an independent arbiter, such as a court of law.

A problem exists if either party has a means of substituting a different hash value during the signing of the agreement. If a hash value from a different agreement is substituted then fraud is possible. If it can be demonstrated that the agreement has been falsified it may be possible to seek compensation or void the agreement.

One method of addressing this issue is to take a secure hash value of the agreement and store it as a witness when the validity of the agreement is doubted. Should A and B separately keep this information it can be used to prove that the agreement has not been altered by either party.

A secure hash function takes a digital representation of a document and subjects it to an algorithmic process to produce a much shorter digital representation, typically 128 bits. Desirable properties of the secure hash algorithm are 'one to one', 'one-way' and simple processing. The process is 'one-to-one' if any two different documents will never produce the same hash sequence. The output sequence is a 'one-way' function of the input sequence when, while easy to compute, it is computationally infeasible to reverse the algorithm to determine the input sequence.

An electronic signature is a binary sequence that is legally accepted as representing the assent and identity of a person in a transaction. Methods of obtaining an electronic identification sequence exist but rely on the initial identification being performed subject to meeting certain conditions. Methods of initially determining the identity rely on existing documents such as a Driver's Licence with a recent picture or birth certificate.

In figure 4.3, an acceptable agreement is hashed to provide a hash sequence. This hash sequence is used to stimulate the individual Smart Cards to obtain an output sequence which is used as a signature. These signatures and the serial numbers of the

Smart Cards are appended to the agreement and it is hashed again. The final hash value generated is retained by the parties to the agreement.

If the signatures are incorporated into the agreement and hashed without a copy of the signature being kept then it becomes harder to alter the agreement and still find an acceptable hash value. Comparing the agreement hash value and the signed agreement hash value may reveal some information about the agreement and the hashing process when the change (signature) is known.

A party may be able to re-create the agreement hash value, personal signature value and the final hash value and use this information to determine the signature of the other party. This information may be used to reconstruct the signature algorithm used by the other party to permit a false signature to be generated.

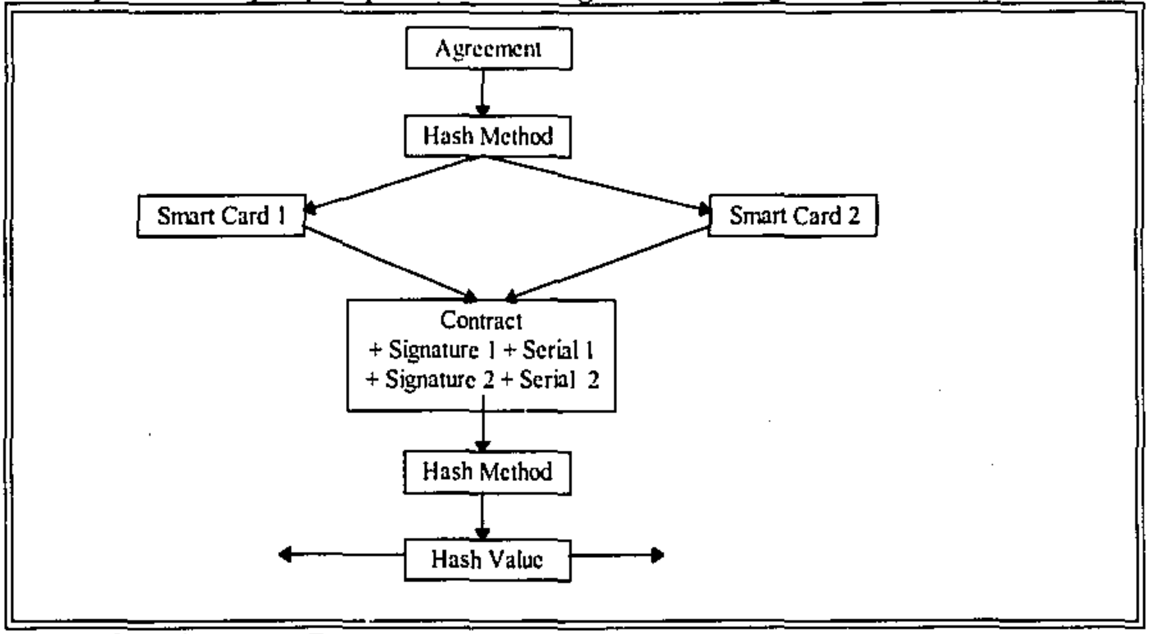


Figure 4-3 Signature Process

All parties to the agreement must be able to verify the hash value produced. A standard hash method can be chosen in the terms of the agreement and all parties can then obtain their own copy from different certified sources. Each party can then use it to verify the agreement before signing and this can detect the substitution of misleading hash values in the transaction.

When a copy of the digital signature of each party to the agreement, including the witnesses, is included in the agreement and hashed, the hash value should show if even a single character of the agreement had been altered. A commonly available method might be MD5, a secure hash function from R Rivest that has been available since 1992. (RFC1321, 1992)

4.3.3. Validation of a Communication between Parties

Communication between remote parties requires identification which can be achieved by possession of some specific piece of information. A password is a subjective identity check where its possession implies approval. Unauthorised possession of the password confers access to the system and security requires a series of different passwords that can be used once and discarded.

Bilateral identity checks using a symmetric cryptographic method requires that each party has the same key information and algorithm to successfully decrypt a message to prove their subjective identity. 'Symmetric algorithms ... are algorithms where the encryption key and the decryption key are the same. .... As long as the communication needs to remain secret, the key must remain secret.' (Schneier,1996, p4) Distribution of the key information is a substantial problem in managing a symmetric cipher system. (Schneier,1996, p177)

If all parties share secret information then identification is implicit in the ability to answer questions about the secret information.

Identity validation could result from the interrogator asking a question about the secret information at random without knowing the answer beforehand. The respondent must supply the answer to the question and the interrogator then generates the correct answer from the secret information and compares the two answers.

If the interrogator does not know the answer beforehand it is more difficult to detect the answer by illegal or unauthorised means. The process must be secure against guessing the answer to a question from knowing previous questions and answers. If the questions are generated by a random sequence it may be very difficult to predict the questions and answer them.

Continuous validation of the communication can be made by periodically interrogating the respondent about the secret information to detect substitution. For example, after a remote log-in to a computer system is made the legitimate user is disconnected and access is assumed by an unauthorised user. The legitimate user may accept this event as a normal system error and not report the problem for some time.

4.3.4. Design Requirements

A design is proposed in figure 4-4 where the Smart Card is configured as a 'one-to-one' function generator which provides a unique binary sequence in response to any unique sequence being supplied to it. This would permit each Smart Card to be used as a signature generator after being assigned to a person or corporation by any acceptable means.

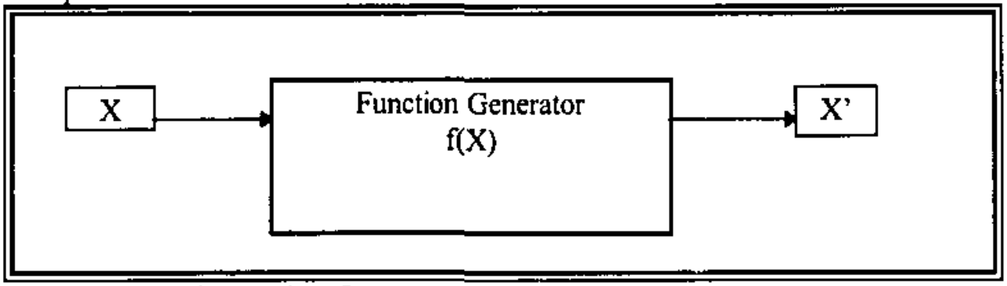


Figure 4-4 Schematic of Proposed Function Generator

To enable a transaction to be remotely validated a copy of the randomiser details on the Smart Card must be kept. A company that issues a Smart Card must keep a copy of the randomiser and selection algorithms for each unique card. Families of cards using the same selection algorithms and / or randomisers can be created to reduce storage requirements.

External storage requirements may be as little as two kilobytes per card which would require approximately 2 Gigabytes for a population of one million cards. However, supporting data may raise this value substantially.

4.4. Proposed Implementation of a Smart Card Function Generator

4.4.1. Introduction

The Smart Card function generator must generate a binary sequence output in response to a binary sequence input. The function must be complex and difficult to predict to provide a measure of security to the information processed.

There is no requirement for an input sequence and output sequence to be of the same length. The function must be 'one-to-one' where each unique input sequence produces a unique output sequence.

This requires that the output sequence is equal to or greater than the length of the input sequence and the algorithm is thoroughly validated wherever possible. An exhaustive search is likely to be computationally infeasible and certification must rely on mathematical proof.

4.4.2. Description of method.

The Randomiser is a random binary sequence stored in a circular buffer able to be accessed at any point and traversed in either direction.

In figure 4.5 the Selection Method specifies a Start Point in the circular buffer to select the first bit and a Selection Set of integers to specify the number of bits to move or discard between each subsequent selection.

A positive value for the move or discard number specifies movement in one direction and a negative value specifies the opposite direction.

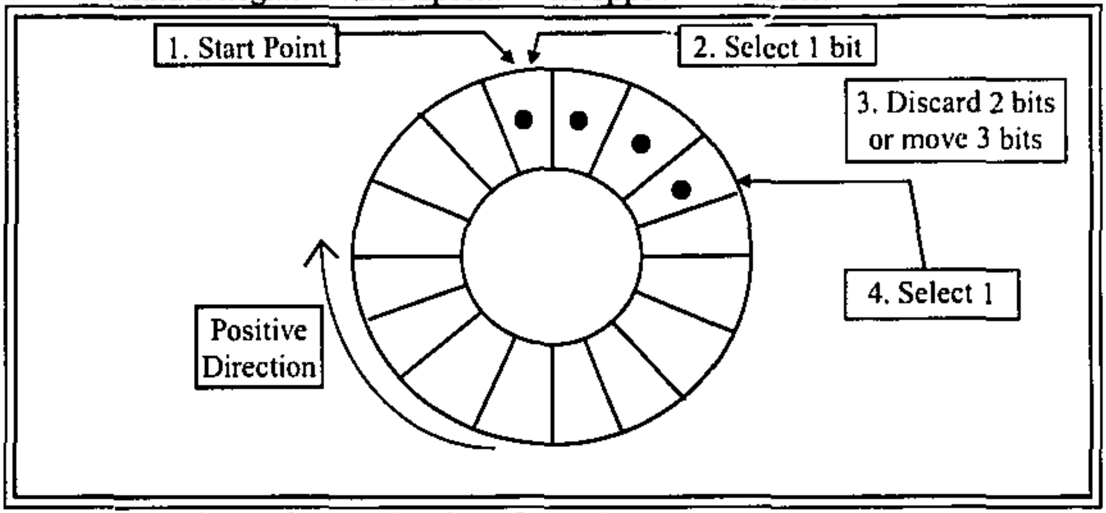


Figure 4-5 Schematic of Selection Method

A Start Point value greater than the size of the buffer bits will be constrained by the modulo division of the Start Point by the size of the buffer. Consequently, there are many ways of specifying the same starting bit. The move or discard values in the Selection Set are also constrained by the length of the buffer.

While permitting an infinite number of ways of specifying the selection of the next bit, there is a limit on the number of unique sequences that may be chosen. With the constraint that no bit in the buffer is ever used twice in each sequence, the size of the problem reduces to the number of combinations of bits taken from the buffer without replacement.

This is achieved by ensuring that the total number of bits selected or discarded is less than the length of the buffer and the direction of buffer traversal is never reversed. Under these conditions no bit is used twice but the option exists as desired.

With these conditions, the maximum number of Sequences, C, is a simple combination of n bits taken from N Randomiser bits and is specified by

$$C = \frac{N!}{n!(N-n)!}$$

Equation 4-1

A combination is defined as an unordered multiset without repetition (LEW,1985, p46)

If it is intended that any bit may be reused by the Selection Method then the number of sequences is much larger as this amounts to selection with repetition. The number of selections, S, of n bits from N bits is specified by

$$S = \frac{(N+n-1)!}{n!(N-1)!}$$

Equation 4-2

A selection is defined as an unordered multiset with repetition. (Lew,1985, p264)

Localised non-randomness in a sequence is defined as any sub-sequence that is strongly patterned. For example, taken in isolation, the sequence '0101010101010' appears predictable and the sequence '101101000000000000101' appears to have an atypical sequence of zeros in the centre while the other sub-sequences appear to be '101'.

It is expected that while each sample taken from a population may exhibit localised non-randomness the characteristics of a large number of samples will exhibit the characteristics, as a group, of the population. For example, taking samples of 4 bits from a random population, the sequence '0000' should occur approximately once every 16 samples.

The statistical concept of the 'Law of Large Numbers' describes where the characteristics of a group of samples will tend to be similar to the characteristics of the population as the number of samples increases.

Briefly, the Law of Large Numbers can be described as

> *"For example, the act may be flipping a symmetrical coin, A may be the event "heads", and it is assumed that the probability of heads, 1/2, is the same from one flip to next. It is also assumed that every trial is independent of (in no way affected by) every other trial. Now after n trials of the act, the proportion of times A has occurred is p. It can be proved (the proof is so difficult there is no point presenting it here) that p gets closer and closer to P(A) as n becomes larger and larger. We can make the proportion of times A occurs as close as we want to P(A), the probability calculated from the sample space, by performing the act enough times. So P(A) tells what will happen in the long run if we actually performed the actions under the conditions laid down above."* (Glass & Stanley,1979, p202)

To ensure a thorough distribution of the samples, the Start Points should be uniformly distributed across the randomiser interval and the Selection Sets should vary in size and content to ensure that all bits are usable.

4.4.3. Constraint on Generating All Possible Sequences

Analysis shows that the largest number of unique samples from the population is a simple combination of Sample bits from Population bits.

Combinations of Size n taken from population of N				Unique Seqs
Sample	N = 2000	N = 4000	N = 10000	2^n
n = 128	$1.3986 * 10^{205}$	$3.857 * 10^{244}$	$8.1243 * 10^{295}$	$3.4028 * 10^{38}$
n = 256	$5.2468 * 10^{330}$	$3.734 * 10^{411}$	$3.0654 * 10^{515}$	$5.2468 * 10^{77}$

Table 4-1 Number of Combinations compared against unique sequences

In Table 4.1, the number of combinations of n bits from N is substantially greater than the number of unique n bit sequences specified in binary, which is given by the limit, 2^n.

Consequently, the total number of bits, T, to be examined would be the number of combinations, C, multiplied by the number of bits, n, in the sample.

$$T = C \times n$$

Equation 4-3

The composition of the population may mean that repetitions of some binary sequences occur and it is considered computationally infeasible to enumerate all the samples or calculate simple statistics such as a frequency distribution for any reasonable size population.

4.4.4. Bremermann's Limit

Enumeration of all the possible samples from any reasonable size population becomes infeasible owing to the amount of bits to be examined. McNeil and Freiberger cite 'Bremermann's Limit' which can be used as an approximation of the magnitude of the task.

Relying on quantum theory, he |Bremermann| first determined how much information the most efficient computer imaginable could process each second. He found that, for each gram of mass, no data processing system - brain, machine, anything- could handle more than twice 10^{47} bits per second.

> *He then asked, "What if a computer were as big as the earth, and it had laboured on a single problem since the earth was born? How many bits could it process?" His answer was 10^{93}, usually called Bremermann's Limit. It is a huge figure - 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000, 000,000,000,000,000,000,000,000,000,000,000,000,000,000.* (McNeil and Freiberger,1993, p166*)*

Representing this number in a binary form:

$\log_2$ (Bremermann's Limit) = $\log_{10}(1 * 10^{93}) / \log_{10}(2)$

$\log_2$ (Bremermann's Limit) = $93 / 0.301029995$

$\log_2$ (Bremermann's Limit) = 308.9393128

Consequently, enumeration of all possible binary sequences is infeasible as Bremermann's limit of 10^{93} can be approximated by 2^{309} .

4.4.5. Distribution of Samples.

Every unique combination from a population can be enumerated by the decimal equivalent of the symbols selected.

In Table 4.2 showing a three bit sequence {A,B,C} there is one way of taking no symbols, three ways of taking one symbol, three ways of taking two symbols and one way of taking three symbols. Each sample can be indexed by the three bit binary sequence from 0 to 7 where each '1' in the sequence identifies the bit selected.

Tallying each sample by its decimal value to determine the distribution of the combinations will characterise a population. Using a population of six bits with a sample of three bits yields 20 different three bit sequences (from equation 4.1) and these samples can be tallied by their binary values from 0 to 7.

In Table 4-3, the resulting totals of each sequence characterise the population.

To specify all the possible randomisers (2^6) and the possible sequences (20) would mean that 1280 possible 3 bit sequences must be enumerated. This can be specified by the pair (I,J) where I is the decimal value of the population and J is the decimal value of the sample. The range and domain of these pairs can be arranged in an array where the

Index	Binary	Sample
0	0 0 0	– – –
1	0 0 1	– – C
2	0 1 0	– B –
3	0 1 1	– B C
4	1 0 0	A – –
5	1 0 1	A – C
6	1 1 0	A B –
7	1 1 1	A B C

Table 4-2 Enumeration of Samples

elements are the tally of each sample.

A population of '000000' would produce 20 sequences of '000' and could be described as (0,0) = 20. A mixed sequence of 1's and 0's would produce a different distribution.

There is an apparent symmetry about the horizontal and vertical axes which would reduce the amount of work in characterising the array.

Decimal	Value	0	1	2	3	4	5	6	7
Value	Binary	000	001	010	011	100	101	110	111
0	000000	20	0	0	0	0	0	0	0
1	000001	10	10	0	0	0	0	0	0
2	000010	10	6	4	0	0	0	0	0
3	000011	4	12	0	4	0	0	0	0
4	000100	10	3	6	0	1	0	0	0
...	...	...	...	...	...	...	...	...	...
61	111101	0	0	0	0	0	4	6	10
62	111110	0	0	0	0	0	0	10	10
63	111111	0	0	0	0	0	0	0	20

Table 4-3 Characterisation of a Sample Space (64,8)

The number of cells in the array is calculated by :

$$A = \text{Columns x Rows} = 2^{N+n}$$

Equation 4-4

where N = number of possible unique sequences and n = number of bits sampled.

In equation 4.4, the number of all possible sequences, A, is increases exponentially for linear increase in the size of the population or sample. Consequently, analysis of large sequences becomes impractical for any reasonable value of N.

4.4.6. Summary

- The proposed method offers ease of use and space efficiency as advantages compared to the existing methods. Additionally, the running speed of the application is considered to be faster than existing block cipher methods and comparable to existing stream cipher methods.
- Starting with the theoretical 'Provably-Secure Strongly Randomised Cipher' (Maurer90) and developmental work (Shaw,1993). The implementation of the proposed method addresses the recommendations (Maurer,1990, p372) relating to broadcasting the randomiser and the difficulty of selecting the bits.
- While the theoretical method is provably secure under certain constraints, the proposed method is provably secure under one constraint only. The constraint is single use of the randomiser which ensures that the proposed method meets the requirements of a 'One-Time Pad'. However, multiple use of the proposed method may be conditionally secure and evaluation of this is beyond the scope of the research.
- Additionally, the theoretical method requires a very large randomiser which is difficult to store and transport.(Menezes et al., 1997, p170)
- The precise description of both methods is covered in the next chapter.

4.5. Construction of Randomisers.

4.5.1. Introduction

The size of randomiser required by the theoretical method is of the order of 1 * 10^{22} bits and algorithmic generation from a smaller sequence may, under certain conditions, be indistinguishable from a complete randomiser produced by a random process.

Non-deterministic sequences cannot be generated a second time even with identical generator setup conditions. Generally, the sequence must be retained and copied as needed which imposes storage and transport constraints on the user. Non-deterministic methods are generally based on physical phenomena.

Deterministic sequences are able to be repeated with the same initial setup conditions.

Consequently they can be reproduced by specifying the algorithm and the setup conditions and are economical to transport and store.

Generation of deterministic sequences include hardware, software and biometric methods.

4.5.2. Non-Deterministic Construction

Methods of generating random binary squences in a non-deterministic fashion are generally based on physical phenomena.

Examples include:

1. elapsed time between emission of particles during radioactive decay;
2. thermal noise from a semiconductor diode or resistor;
3. frequency instability of a free running oscillator;
4. the amount a metal insulator semiconductor is charged during a fixed period of time;
5. air turbulence within a sealed disk drive which causes random fluctuations in disk drive sector read latency times; and
6. sound from a microphone or video input from a camera

(Menezes et al.,1997, p172)

Thermal noise is a form of white noise that is caused by the random thermal agitation of electrons in resistive components.(Young, 1979, p 342) Bassein describes an electron noise amplifier that is economical and simple to construct. (Bassein,1996, p489) Another noise generator can be made by configuring a serial port on a PC and connecting to it a simple circuit that samples the noise input from a radio. (Appendix 9.7) Additionally, a circuit to sample 256 values through a PC parallel port is readily implementable when connected to a suitable random source. (EA,1997, p51)

'Sampling diode shot noise' and time between decay of radioactive particles can be used as well as the Very Large Scale Integration (VLSI) implementation of a Metal Insulator Semiconductor Capacitor. Comparison of the charge between two capacitors can produce a random result. (Agnew,1987, pp77-81)

However, small random sequences up to 1000 bits can be produced by methods such as Keystroke timings, hardware functions and cryptographic conversion. (Roberts,1995, p105)

4.5.3. Deterministic Construction

Deterministic methods include hardware, software and hybrid systems.

Hardware methods include the Linear Feedback Shift Registers (LFSRs) which are described (Schneier,1996, pp369-379) and analysed. (Barker,1984) Variations on the theme are described by Schneier (Schneier,1996, pp380-388 ; Zeng et al.,1991, p8-16)

Software Methods include Linear Congruence Generators and implementations of a LFSR. (Knuth,1973, pp9-25; Schneier,1996, pp369-362)

Processes upon which software random bit generators may be based include:

1. the system clock;
2. elapsed time between keystrokes or mouse movement;
3. content of input/output buffers;
4. user input; and
5. operating system values such as system load and network statistics.

(Menezes et al.,1997, p172)

Many deterministic methods are cyclic in that the output sequence is likely to repeat during a long run. The length of the cycle is an important characteristic in measuring the performance of a pseudo-random sequence generator.

Methods of 'De-Skewing' the sequence can include applying a secure hash function to pseudo-random sequences to 'distil the (true) random bits from the sampled sequences.' (Menezes et al.,1997, p172)

4.5.4. Hybrid Construction

The proposed method is a hybrid which is not as space efficient as deterministic methods but has the advantage of being able to generate and store very large sequences. The use of random sequences rotated according to some simple algorithm may be indistinguishable from a sequence generated from a random process under some circumstances.

An operation, ROT(), rotates a sequence of symbols without loss of data and is used to generate deterministic changes. With a Basic Sequence of '123', ROT(1) gives '231', ROT(-1) on '231' gives '123' and ROT(3) on '123' gives '123' which is identical to ROT(0).

The three unique sequences, 123-231-312 (enumerated from 0 to 2), are concatenated into a Complete Composite Sequence containing one example of every rotation. Concatenation of these sequences can be performed 3! or six ways and each Complete Composite Sequence can be concatenated to produce a much larger sequence.

The rotation operation preserves some characteristics of the Basic Sequence. If P(0) denotes the probability of obtaining a zero for any one bit sample of the Basic Sequence then P(0) is the same for the Rotated Basic and Composite Sequence.

A concern would be whether it is practical to determine the Basic Sequence and rotation order to permit the reconstruction of the complete randomiser. Two methods are practical : inspection and analysis.

Inspection detects obvious similarities by simple observation and is limited by the characteristics of the observer. It may be impractical to monitor 1000 bit subsequences in a larger sequence. Analysis is any method that applies an algorithm which produces a result for evaluation. For example, statistical analysis of sequence characteristics or a specific algorithm that halts on a valid solution.

If a contiguous sequence of bits is selected from the Composite Sequence then there may be enough information to determine the Basic Sequence, however, selecting individual bits widely dispersed throughout the Complete Composite Sequence may make this difficult.

If an incomplete Composite Sequence is analysed, the Basic Sequence and rotation sequence may not be readily determined and it may be impractical to determine by inspection or analysis whether the sequence is Random or Composite

The randomiser may be constructed for complexity with more than one Basic Sequence, rotated according to a predetermined selection of values to make analysis more complex. For example, using sequences of different length denoted by A,B,C ...,Z a Complex Composite Sequence can be described as A17, Z123, B6 ... Y19.

A Complete Composite Sequence created with this method could meet Maurer's stated requirements with a Basic Sequence of $1 * 10^{11}$ bits or $1.25 * 10^{10}$ bytes. The information necessary to construct this randomiser is the Basic Sequence and the sequence of numbers identifying the order and rotation of the concatenation operation and this information is much smaller than the Complete Composite Sequence.

A concise description of the method is covered in chapter 5.

4.5.5. Summary

- ❑ Randomisers may be of deterministic, non-deterministic or hybrid construction.
- ❑ Construction of very large randomisers raises difficult generation, storage, search and transport issues.
- ❑ A deterministic generator must produce output sequences that are indistinguishable from non-deterministic sequences.

4.6. Calculation of Large Factorials and Exponentials

4.6.1. Introduction

Initially, the generation of large exponential and factorial numbers provided an understanding of the accuracy achievable when dealing with very large numbers of the order of 10^{300}. However, being able to specify and generate a large sequence of digits was of some use in implementing the developmental algorithm. For example, in appendix 11.8, 2^{1000} =
107150860718626732094842504906000181056140481170553360744375038837035
105112493612249319837881569585812759467291755314682518714528569231404
359845775746985748039345677748242309854210746050623711418779541821530
464749835819412673987675591655439460770629145711964776865421676604298
31652624386837205668069376.

These sequences provide long but easily specified sequences of digits for use in selection algorithms

The calculation of factorial numbers constitutes some challenges due to the exponential nature of the numbers produced and the errors occurring due to the nature of the calculations.

Operations using logarithms may underestimate the number due to truncation caused by the method of generating the logarithms. For example, $\log_{10} 10$ is 1 exactly, but, $\log_{10} \Pi$ is irrational as Π is irrational and any representation of Π will incompletely estimate it. However, where Π is the basic unit in which all other quantities are expressed any gain in accuracy may be lost in converting to other units. For example, $\omega = 2\pi$ is more precise than the decimal notation $\omega = 6.28318$.

4.6.2. Methods Used

Three methods of calculating large factorial numbers were considered. Calculations use Stirling's Approximation to Large Factorials, computer based sequential addition of logarithms and integer methods.

4.6.2.1. Calculation using Stirling's Approximation :

Stirling's Method of calculating large factorials is a well known method in use for many years. (Spiegel, 1972, p 114)

$$N! \approx \sqrt{2\Pi N} * N^{N} * e^{-N}$$

Equation 4-5

$$\log N! \approx 1/2 \log(2N) + 1/2 \log(\Pi) + N \log N - N \log e$$

Equation 4-6

A logarithm of an irrational number is a source of round-off errors. All logarithms are real numbers excepting the comparatively few integer logarithms. Unless the exact logarithmic value for a number is calculated and used, any calculation relying on logarithms may not estimate the correct result owing to 'rounding errors' or truncation.

4.6.2.2. Calculation using Sequential Addition of Logarithms:

Sequential addition of logarithms uses the additive properties of logarithms to perform the multiplications. The calculation model is

$$\log N! \approx \sum_{i=1}^{N} \log i$$

Equation 4-7

Addition of logarithms avoids the problems of multiplying real numbers and is relatively fast. The largest number able to be calculated is limited by the integer component of the logarithmic sum. For example, the logarithm 1234.56 evaluates to $3.6307 * 10^{1234}$ and in 32 bit numbers, the largest exponent would be 2^{32} -1 and consequently, the largest number that can be calculated by this method is approximately $9.9 * 10^{4294967295}$

Logarithms generated by software methods are subject to errors and to extend the accuracy of the calculations more accurate logarithmic values are needed. Generation of logarithms requires the use of the geometric mean which needs a square root function and may give results with limited precision due to the nature of calculating the square root.

4.6.2.3. Calculation of Logarithms

The algorithm for generating a logarithm of a number (McLeish, 1991, pp171-173) is

- Start with any two numbers and their known logarithms
- For a new and unknown logarithm, work out the arithmetic mean of the two earlier logarithms
- For the unknown number work out the geometric mean of the numbers

The arithmetic mean is calculated by summing the two numbers and dividing by 2.

The geometric mean is calculated by multiplying the two numbers together and taking the square root.

For example: GM = geometric mean, AM = arithmetic mean

Numbers		Logarithms	
100	1000	2	3
GM = 316.22766		AM = 2.5	
100	316.22766	2	2.5
GM = 177.827941		AM = 2.25	

100 177.827941 2 2.25
GM = 133.3521432 AM = 2.125

(McLeish, 1991, p173)

The logarithm of 133.35214432 is 2.125.

Munro gives an algorithm for calculating Square roots:

Algorithm 2.8 to calculate to some given accuracy the positive square root of a positive real number.

```
Input:  k, real; k > 0.
(we wish to find the square root of k)
e, real; e > 0.
Output:         y, real.
(y is the final approximation to $\sqrt{k}$)
method:         give x a value; any positive number will do.
$i \leftarrow 0$  (give i the initial value 0)
repeat
        $x \leftarrow (x + k/x)/2$        ( x is the current approximation to $\sqrt{k}$)
        $i \leftarrow i + 1$
until x is close to $\sqrt{k}$
$y \leftarrow x$
```

What does close mean?
x is close to $\sqrt{k}$ may be refined to
(|x^2 - k| $\le$ e) is true
or (|x - the previous x| $\le$ e) is true
(Munro, 1992, p36)

It can be seen that calculating square roots requires addition and division with the root being a real number.

4.6.2.4. Calculation by Integer Methods

Large factorials can be obtained by performing iterative integer operations on numbers and storing the results. These numbers are considered to be more accurate than either of the preceding methods.

Integer calculation of large exponentials and factorials requires substantial computer time and memory. Using a representation of integers that requires one byte per digit, at least 4N bytes of storage and working space are needed to calculate and store $X_0 = X_1 * X_2$. For example, $X_1 = 1234$ and $X_2 = 1001$ gives $X_0 = 1235234$ which shows that two four digit numbers multiplied together produce a seven digit number totalling 15 digits in all.

Digits are represented as one byte short integers though only values of 0 to 9 are used and the numbers are stored in reverse order in a sequential file. Multiplication occurs from the least significant digit at the beginning of the file by sequential file access and any 'carry' digits are appended to a file. This uses conventional file access routines. More space efficient methods such as Binary, Binary Coded Decimal (BCD), Octal or Hexadecimal may be time consuming to process.

A typical hard drive may contain 2.1Gigabytes of data and consequently limits the size of the largest number processed with it to approximately 525 MegaBytes or x.xx * $10^{525000000}$.

4.6.2.5. Calculation of Large Exponentials of the Form 2^n

Large exponentials are easily calculated using the same iterative integer methods. An approximate check on the accuracy of the numbers generated was performed by comparing the value given by logarithms and the value given by direct calculation. For example, in Appendix 11. calculating 2^{1000} by logarithms gives 1000 * log 2 which is 301.029995664 or 1.0715 * 10^{301}

4.6.3. Summary

- The methods listed have differing characteristics which may indicate uses.
- The fastest but least accurate method is Stirling's Approximation.
- Sequential summation of logarithms is slower but much more accurate. It is limited by the representations of real numbers on the computer being used. Additionally, different data types may alter the precision and accuracy of the numbers generated.
- Direct Calculation is the most accurate method but is very slow. The major requirements are memory and this limits the speed by requiring the operating system to swap large amounts of data between files and RAM. Use of a RAMDISK to speed up the process is limited by the amount of usable RAM.
- Calculation of large factorials and exponentials files can be expedited by use of stored pre-calculated numbers. For example, to calculate 1001!, the file 1000! Is multiplied by 1001 and the results stored. The results are included in the Appendix 10.8.

4.7. Test 'T' for Randomness

4.7.1. Introduction

A method called the 'Test-T' was implemented as an aid to estimating 'randomness' in a binary sequence generator. The test uses the established idea of measuring the relative probability of the occurrence of bit patterns of varying lengths and comparing them against a theoretical value (Bassein, 1996, p483). Implementation is relatively easy however, interpretation of the results may rely on substantial statistical knowledge.

This test is similar in construction to the 'Universal Test' (Maurer,1990) that calculates the average logarithm of the time interval between successive occurrences of the same subsequence.

The results of both tests can then be used to assess the performance of the Random Sequence Generator.

In Test-T the theoretical value for each bit pattern is the probability that each sequence would appear assuming a random sequence of almost infinite length is examined.

The decision to take overlapping or non-overlapping samples of the sequence is important. Overlapping sequences simulate a binary source with memory, where the output bit is dependent on the preceding bits. Non-overlapping sequences simulate a random number generator where each group of bits represents an integer in binary form.

The concept of the 'Test-T' leads to the establishment of a classification system which can identify important characteristics of random or pseudo-random sequence generators.

4.7.2. Methods Used

4.7.2.1. Universal Test

An implementation of the Universal Test is available in the Crypt-X package and a simple implementation in Pascal is provided in the appendices.

The Universal Test (Maurer,1991, p89) is implemented as an array of integers where the index to the array is the binary value of the sequence tested. The array is cleared and a selection of samples from the generator is taken and the array is pre-loaded with the time each binary value appeared. The time unit is defined as one sample and consequently the number of samples taken denotes the elapsed time.

The pre-loading for a number of samples ensures that there is a reasonable chance that all the array components are non-zero. The test proper uses the decimal value of the sample as an index to the array which stores the previous time of occurrence of the value. Comparison of the time of the sample with the previous time in the array permits the logarithm of the difference between them to be stored in an accumulator. The logarithm is summed until the end of the test when it is averaged by the number of samples taken. This average logarithm is generally close to n-1 bits for a random source.

In figure 4.6, the count prompts the generator to produce a binary sequence. The value of the binary sequence is used to index the array to store the count until the pre-loading is complete.

When the test starts, the number records the amount of samples tested. Each binary sample value points to the array where the count of the previous occurrence is stored. The previous count is subtracted from the current count and the logarithm of the result is stored in an accumulator. On completion of the test, the value in the accumulator is divided by the number of samples tested to produce the average logarithm.

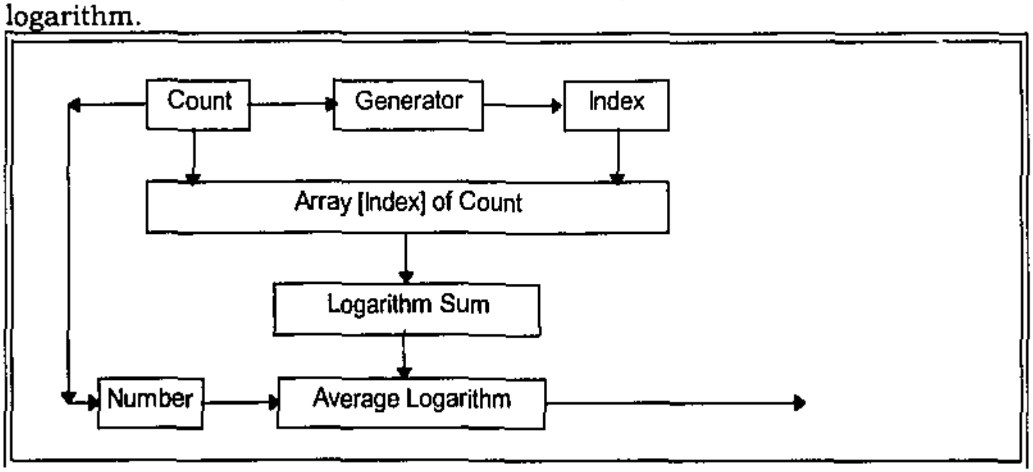


Figure 4-6 Schematic of Universal Test

The average logarithm is the average 'distance' between occurrences of a particular binary sequence. If the generator is random and the samples are uniformly distributed then the average logarithm approximates n -1.

For example, using 8 bit samples the array will be indexed between 0 to 255. Eight bit samples from the generator are used to index the array to store the count. In a uniformly distributed sample from a generator a sample value of '0' is equally likely to followed by '0' or any value to '255'.

The average distance between these two equally likely values is 128. $\text{Log}_2(128)$ = 7 which is n - 1. Any persistent or substantial difference between the actual and predicted logarithms can be used to characterise a generator.

4.7.2.2. Test-T

In Figure 4-7, a simple model to permit a basic description of randomness was developed. Instances of each bit pattern are recorded by dedicated counters. The assumption is that random sequences of binary digits of extreme length are available.

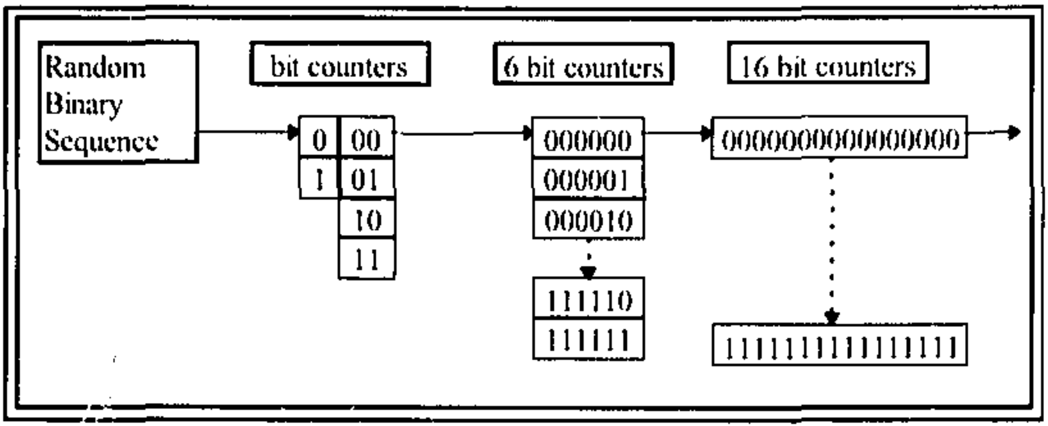


Figure 4-7 Schematic of Theoretical Test-T for Randomness

In this population of nearly infinite size each pattern of binary digits will be well distributed throughout the population and the probability, P, of each pattern will be specific. (Peterson,1998, p179)

While a sequence may produce a probability $P(0) = P(1) = 0.5$ for single bit samples it may not produce a probability result of 0.25 for each two bit pattern.

Using non-overlapping samples:

$S = \{10011001\} \Rightarrow P(0) = P(1) = 0.5$

$S \Rightarrow \{10\}\{01\}\{10\}\{01\} \Rightarrow P(01) = P(10) = 0.5, P(00) = P(11) = 0$

These results can be described as, 'the sequence passes T^1, a test for one bit patterns, but fails T^2, a test for 2 bit patterns' and there are insufficient bits to evaluate T^3, a test for three bit patterns.

Using the Test-T it is not possible to perform a complete range of tests and consequently no sequence can be considered truly random as it is possible for a sequence to pass test T^k but fail test T^{k+1}.

In Table 4-4, an indication of the resources necessary is the total number of accumulators needed.

Accumulators Required

n	2^n	Sum	2^{n+1}	$2^{n+1}-2$
1	2	2	4	2
2	4	6	8	6
3	8	14	16	14
4	16	30	32	30
5	32	62	64	62

Table 4-4 Accumulators needed for Test T

Table 4.4 shows that for each number of bits, n, the number of individual accumulators needed is 2^n. For five bit tests the number of accumulators needed is the sum of the powers of 2, which equates to $2^{n+1} - 2$. The exponential increase in the number of accumulators needed limits the amount of testing that may be performed.

N	$N * 2^n$	Bits Required
2	$2 * 2^2$	8
32	$32 * 2^{32}$	$1.374 * 10^{11}$
64	$64 * 2^{64}$	$1.1805 * 10^{21}$
128	$128 * 2^{128}$	$4.3556 * 10^{40}$

Table 4-5 Number of Bits required to perform Test T^n

Additionally, in Table 4-5, a limitation on the Test-T is the amount of bits necessary to perform the test with any confidence.

Using non-overlapping samples, the minimum number of bits to test is $n * 2^n$ to obtain a valid measurement. For example, using 2 bit sequences we need to test at least $2 * 2^2$ or eight bits. This will be the minimum possible to obtain a probability of 0.25 for each two bit sequence though testing a million bits may improve the precision of the result. The exponential increase in the number of bits needed ensures that it is not practical to perform the Test-T for large values of n.

Using the Test-T, a classification system based on the size of the sequence tested and the number of unique symbols available is possible.

A proposed standard notation is described below. The superscript, a, defines the largest test evaluated, the subscript, b, the number of unique sequences available from the generator, X is the enumeration of a specific output sequence and m is the number of symbols of generator memory :

$${}_{m}\mathrm{T}_{b}^{a}\,X$$

A ${}_{1}\mathrm{T}_{2}^{1}$ generator is a binary source with memory of 1 bit, that is, the output bit is dependent on the previous bit. Further discussion of the classification scheme is Section 4-8.

Results from testing a fair coin can be listed as: T_2^1 H = 0.497 and a fair die, T_6^1 3 as 0.1659.

For example, a single, fair, six-sided die would be classified as a T_6^1 generator as it produces only 6 unique symbols, one at a time. Additionally, two fair dice, a T_6^2 generator passes a test for two symbols, that is, every pair of symbols is equally likely. It is noted that the two six-sided dice produce 6^2 or 36 different sequences as they are independent of each other and would pass T_{36}^1 , a test for individual symbols from an alphabet of 36.

A fair coin tossed in conjunction with a fair die to obtain 12 unique pairs of symbols would meet the requirement of T_{12}^1 , implying that the product of the notation subscripts can be summed. For example $\mathrm{T}_6^1 * \mathrm{T}_2^1 = \mathrm{T}_{12}^1$ giving 12^1 or twelve unique symbols.

4.7.2.3. Results

An implementation of the Test-T produced the following results for runs of various length. In Table 4-6, four runs were processed of 1000, 10000, 100000 & 1000000 bits. The one bit and two bit patterns were generated and the results tabulated. The generator appears to prefer 1's slightly more frequently than 0's during shorter runs. Values below the expected value are underlined.

Bits	0	1	Average
1000	0.49200000	0.50800000	0.50000000
10000	0.49870000	0.50130000	0.50000000
100000	0.49948000	0.50052000	0.50000000
1000000	0.50016100	0.49983900	0.50000000

Table 4-6 Single Bit Results

In Table 4-7, the two bit patterns show similar preferences, however, the long runs seem to favour '00' compared with shorter runs which favour ones.

Bits	00	01	10	11	Average
1000	0.2380000	0.25400000	0.25400000	0.25400000	0.25000000
10000	0.2478000	0.25090000	0.25090000	0.25040000	0.25000000
100000	0.2495900	0.24989000	0.24988000	0.25064000	0.25000000
1000000	0.2502500	0.24991100	0.24991100	0.24992800	0.25000000

Table 4-7 Two Bit Results

A major concern with this simplistic test is the setting of limits for the values and interpretation of the results. These must be determined carefully to permit a classification to be made.

4.7.3. Summary

- The Universal and Test-T are easily implemented in software, however, interpretation of the results may require additional information and testing.
- A classification which is able to clearly identify a random sequence generator by its output characteristics may make specification and selection of a random sequence generator efficient.
- Methods of assessing random generators for classification are not dealt with in detail.

4.8. Classification Scheme

4.8.1. Introduction

A method of specifying clearly the characteristics of random symbol generators is proposed as an aid to selection and classification of generators. The classification method must simply identify important characteristics for random generators to permit a specification to be made that does not rely on often misunderstood statistics.

Ideally, a system designer should be able to specify a type of random sequence generator and permit the final selection to be made during implementation.

An introduction to the concept of 'Combinational Calculus' using the proposed classification system is provided to describe the common practice of combining random generators into a composite generator. This work is to be developed further and validated and consequently is neither complete nor rigorous.

It can be shown that generators can be combined using logical operations and the description can provide an operator with a summary of the composite generator's characteristics. It is possible that this concept can be extended and used to describe aspects of complex systems which contain random sequence generators.

4.8.2. Explanation of Classification Method

Two fair six-sided dice D1 and D2 are defined as D1 = {a,b,c,d,e,f} and D2 = {g,h,i,j,k,l}, both of which are classified T_6^1 and two fair coins are defined as C1= {m,n} and C2 = {o,p} both of which are classified T_2^1 . The 'T' specification means that the sequences produced by each artefact are random.

For example, the sequence '01011010', while obviously patterned has the probabilities of P(0) = P(1) = 0.5. This does not necessarily mean the generator of the sequence is random but it passes test T_2^1. However, the probabilities for T_2^2 are P(00) = 0, P(01) = 0.5, P(10) = 0.5 and P(11) = 0 and this means that the generator 'fails' test T_2^2 . Passing test T_2^1 is implicit in passing test T_2^2 and failing T_2^2 may mean that the sequence examined is insufficient to determine the result.

Consequently, no test can be certain that a sequence is random.

Logical operations may be used to describe the use of more than one generator to produce specific results. Being able to describe clearly the effects of combining two or more generators makes it easy for decisions to be made without substantial statistical knowledge.

4.8.3. Classification Relationships

{ Θ } = the empty set. $\wedge$ means logical and, $\vee$ means logical or

Logical NOT negation

$C1 = \{m \vee n)$ if $C1 = m$ then NOT $C1 = n$ if $C1 = n$ then NOT $C1 = m$
$D1 = \{ a \vee b \vee c \vee d \vee e \vee f \}$ If $D1 = a$ then NOT $D1 = b \vee c \vee d \vee e \vee f$

Logical AND *
$D1 \wedge D2 = \{a\wedge g \vee a\wedge h \ldots \vee b\wedge g, \ldots \vee f\wedge l\} \Rightarrow T_6^1 \wedge T_6^1 = T_6^1 * T_6^1 = T_{36}^1$
$C1 \wedge D1 = \{ m\wedge a \vee m\wedge b .. \vee n\wedge a \ldots \vee n\wedge f\} \Rightarrow T_6^1 \wedge T_2^1 = T_6^1 * T_2^1 = T_{12}^1$
where $a\wedge g$ is considered to be one symbol.

$D1$ AND NOT $D1 = \{ (a\wedge(b \vee c \vee d \vee e \vee f)) \vee \ldots \vee (f\wedge(a \vee b \vee c \vee d \vee e)) \} \Rightarrow \{ \Theta \}$
$= \{a\text{-}a \wedge b\text{-}b \wedge \ldots \wedge f - f\} = T_6^1 - T_6^1 = T_0^1 = \{ \Theta \}$

$D1/D1 = \{a/a, b/b \ldots e/e, f/f\} = \{1,1,1,1,1,1\} = \{1\} =$ Identity set.

$D1 \wedge (D2 - D2) = \{a+g-g, \ldots, f+g-g\} = \{a,b,c,d,e,f\} \quad T_{12}^1 - T_6^1 = T_6^1$

$D1 \wedge D2 \wedge C1 = T_6^1 * T_6^1 * T_2^1 = T_{72}^1$

$D1 * D2 / D2 = T_6^2 / T_6^1 = T_6^1 = \{ag,ah,ai,\ldots fj,fk,fl\} / \{g,h,i,j,k,l\} = \{a,b,c,d,e,f\}$

$D2/(D1/D1) = D2/1 = D2$

Logical OR
$D1 \vee D2 = \{a \vee b \vee .. \vee l\} \Rightarrow (D1 \wedge C1) \vee (D2 \wedge \text{NOT } C1) \Rightarrow (T_6^1 \wedge m) \vee (T_6^1 \wedge n) =$
$T_6^1 \wedge T_2^1 = T_6^1 + T_2^1 = T_{12}^1$
equivalent to using a fair coin, C1, to specify either D1 or D2 in each throw.

$C1 \vee D1 = \{m \vee n \} \vee \{a \vee \ldots \vee f\} \Rightarrow (D1 \wedge C2) \vee (C1 \wedge \text{NOT } C2) \Rightarrow (T_6^1 \wedge m) \vee (T_2^1$
$\wedge n) = T_8^1$
equivalent to using a fair coin, C2, to specify either D1 or C1 in each throw.

x' means that this term is NOT x, that is, not included. In any set of two symbol terms, one symbol terms are not part of the set.

each term can be considered expanded, a => [a,b',c',d',e',f'] and b => [a',b c' d' e' f']
then ag => [ag,b'g,c'g,d'g,e'g,f'g] = ag
ag/g => [ag, b'g, c'g, d'g, e'g, f'g] / g => [a,b',c',d',e',f'] => [a]

$(D1 \wedge D2)/D2 = \{ag,ah,ai,\ldots fj,fk,fl\} / \{g,h,i,j,k,l\} = \{a,b,c,d,e,f\}$
$T_{36}^1 / T_6^1 = T_6^1$

4.8.4. Summary

□ Classification schemes for random symbol generators permit a standard classification of the output of either single or composite generators.

□

□ Construction of a composite generator can be specified from two or more different generators according to simple rules.

4.9. Visual Testing of Binary Sequences

4.9.1. Introduction

Assessment of the characteristics of a random bit generator may require substantial test preparation and assessment of substantial amounts of data. An effective method of detecting faulty generators is required which should be easy and efficient to apply.

Graphical methods are proposed for initially assessing a random bit generator. After preliminary assessment specific statistical methods can be used.

A generator may have a 'memory'. This means that the output is dependent on a number of previous output symbols. For example, a 'binary memoryless source' produces statistically independent bits and a source with a memory of one will produce bits dependent on the previous bit. Memory in a generator means that overlapping samples are taken.

Non-overlapping samples mean the generator output is being tested for statistical independence. For example, a three bit generator with overlapping samples is a binary source with memory of two bits and non-overlapping samples means it implements test T_8^1 for three bit sequences.

This simple graphical method may show some characteristics of the generator but any test for randomness must meet stringent requirements. The graphical methods were confined to whole pixel values used to eliminate the need to perform geometric calculations such as Sine and Cosine with the attendant floating point calculations and round-off errors.

4.9.2. Methods Used

4.9.2.1. Single Bit Sequences.

This test applies to binary memoryless sources, defining an array of 2k + 1 integers to plot the output characteristics of a single bit generator. All array values are initialised to zero and the array index is set to the central position, k +1. From this position, the array index is incremented if the randomiser bit is a '1' and decremented if the bit is a '0'. After modifying the index, the integer in the array position is incremented and the run is terminated when the index is moved outside the array.

In figure 5-8, an individual trace represents a single test run on the same generator and the cumulative totals for each array position are kept. The cumulative values describe a line which encloses an area between the horizontal axis and the trace. While some traces appear to follow an approximate bell curve distribution, the center of the area under the line may be to one side of the centre line.

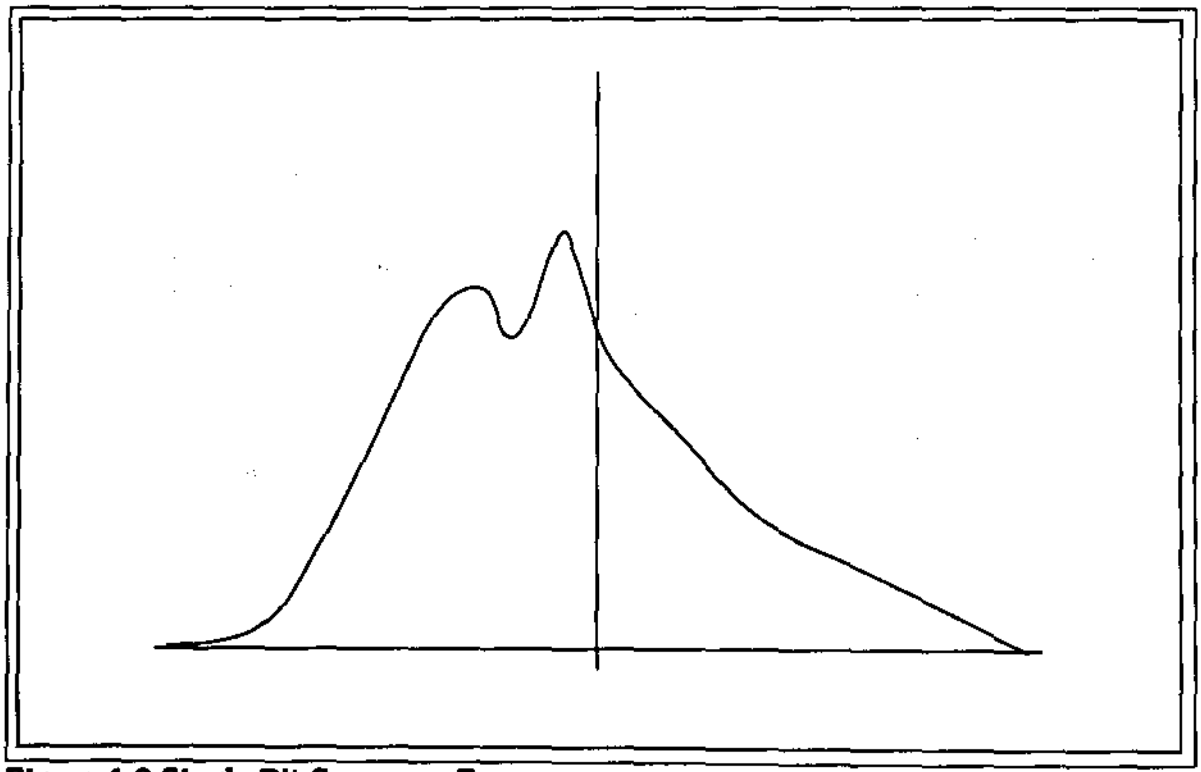


Figure 4-8 Single Bit Sequence Trace

Randomly distributed symbols should be evenly, if not symmetrically, distributed about the centre of the array. A center line consistently to one side of the central axis may be considered evidence of a preference by the generator for a particular value. Additionally, should the trace lack symmetry then this may be considered a characteristic of the generator.

Example for memory-less single bit generators.

For K = 3, an array is defined where X denotes the central position. Starting from this position move to the right if the bit is a '1' and left if the bit is a '0' and increment the count in each array position on entry.

A. Random sequence = {}
0000000
123X321

B. Random sequence = {0101}
0022000
123X321

C. Random sequence = {01010011001}
0263000
123X321

D. Random sequence = {010001100100}
3431000
123X321

In example D the final 0 moves the trace outside of the array and stops the process. It is noted in examples C and D that the digit '0' is slightly more numerous than '1' and the trace shows a preference for that side.

If the values were graphed it would show an approximate bell-curve with the centre of the curve being displaced in the direction of the preferred digit. Results can be seen in Appendix 9-20.

4.9.2.2. Mapping considerations of Multi-bit generators

The mapping of multiple bit sequences to pixel values can be made without trigonometric values such as the sine and cosine of an angle. The method encodes the directions in binary format to ensure that no data is lost when converting the random sequence to the graphical environment.

In figure 4.9 each of the directions from the origin '+' is the decimal representation of the binary sequence from 0 to 2^n-1.

28	29	30	31	0	1	2	3	4
27								5
26		14	15	0	1	2		6
25		13	7	0	1	3		7
24		12	6	+	2	4		8
23		11	5	4	3	5		9
22		10	9	8	7	6		10
21								11
20	19	18	17	16	15	14	13	12

Figure 4-9 2,3,4 and 5 bit direction encoding

The center is defined by the '+' symbol and the 2 bit sequence directions by 0,2,4,6 adjacent to the '+'. Three bit sequences are enumerated by 0 - 7 adjacent to the '+' sign.

In Table 4-8, the size of the square of pixels increases by a binary exponential factor. It can be seen that the difference between 3 bit and 4 bit pixel squares is 2. Between 4 bit and 5 bit pixel squares the difference is 4 and so on.

Index	No. Of Bits	directions	pixel square	difference
1	2	4	2	1
2	3	8	3	2
3	4	16	5	4
4	5	32	9	8
5	6	64	17	16
6	7	128	33	32

Table 4-8 Differences in Pixel squares

It is possible to calculate the size of the pixel square side, V, necessary for n bit resolution by the following formula.

$$V = 2 + \sum_{i=3}^{n} 2^{n-3}$$

Equation 4-8

For an n = 8 bit sequence, the pixel square size = 2 + 1 + 2 + 4 + 8 + 16 +32 = 65 pixels. This pixel square would have two sides of 65 pixels and two sides of 63 pixels totalling 256 pixels. The line representing the direction of the trace can be drawn as a line segment 33 pixels long starting from the origin to the periphery.

Algebraic methods using the known relationship of the numbers of pixels in the sides of the pixel square can be used where two sides are X pixels long and two sides are X - 2 pixels long. The required number of pixels in the pixel square will be two to the power of the number of bits examined.

$$\begin{aligned} 2^n &= 2x + 2(x-2) \\ 2^n/2 &= x + x - 2 \\ 2^n/2 + 2 &= 2x \\ 2^n/4 + 1 &= x \end{aligned}$$

Equation 4-9

For example, 8 bit samples give

$$256 = 2x + 2(x-2)$$
$$65 = x$$

Limitations include the size and resolution of the screen, for example, a screen of 480 * 640 pixels would be able to support a target of a maximum of 480 * 480 pixels and starting from the origin only 7 * 33 pixel line segments would be needed to reach the rim of the target set at a radius of 231 pixels.

Movement size is the integer value of the square size divided by 2.

$$M = \rfloor x/2 \lfloor$$

Equation 4-10

For example, a screen of 640 by 480 pixels would be able to support a square target of a maximum of 479 * 479 pixels. Using a '9 by 9' square, and starting from the origin, only 27 steps are needed in a specific direction to cross the rim of the target set at a radius of 231 pixels.

This method of graphing the output of the Generator is commonly described as the 'Drunkard's Walk', where the notional inebriate takes a step away from a point in a random direction as often as required. The trace produced represents the path taken and characterises the randomness of the directions chosen. The largest binary sequence that can be plotted is limited by the size of the target and the size of the steps.

For much larger binary sequences, the inebriate takes one step away from the origin and is then magically transported back to the origin to take a second step. After a number of iterations, the steps of the inebriate are plotted by making lines to map direction to the screen. This display of the rays traced would characterise the generator.

4.9.3. Summary

- Visual testing methods permit rapid characterisation of a variety of sequence generators.
- Evaluation is limited by the amount of information generated by the test.
- Very large numbers of bits may be tested very quickly.
- Results are listed in appendix 9-20

4.10. Simulation of Drunkard's Walks

4.10.1. Introduction

A 'Drunkard's Walk' is a mathematical paradigm to explain the output of a random sequence generator with the behaviour of a mythical inebriate. The steps taken by the inebriate, where direction is defined by the random sequence, are traced for analysis.

A simulation of a Drunkard's Walk was used as a method of detecting the characteristics of random sequence generators by observing the trace produced from plotting the output of the generator as steps in a Drunkard's Walk. Initially, the directions were encoded for 2 bit, 3 bit and 4 bit sequences and the behaviour observed. Directional preferences in the sequences may be observed, that is, the graph may show a preference for a particular side of an axis through the origin.

In the 'Drunkard's Walk' paradigm the drunk walks away from the origin taking each step in a random direction. The average distance d_a travelled by the drunk after a number of steps, N, of length, λ, is given by

$$d_a = \lambda\sqrt{N}$$

Equation 4-11

(KAYE,1993, p54)

Solving for N, the number of steps needed to travel the average distance,

$$N = \left(\frac{d_a}{\lambda}\right)^2$$

Equation 4-12

The average number of steps needed to travel a particular distance is the square of the average distance divided by the length of step. For example, with $d_a = 200$, and $\lambda = 1$, N is 40000. If the average number of steps differs substantially from this value it may indicate that the generator is biased or not random though this may take many runs of the generator to establish. Kaye mentions 'Many random walks are required before the measured average distance converges on the theoretically predicted value' (Kaye,1993, p55)

4.10.2. Concerns in Simulation of Drunkard's Walks

4.10.2.1. Direction

The selection of direction not magnitude of the steps poses problems when simulating the trace on a computer. Digitisation of direction would introduce quantisation error when using real values. For example, in the real world, an itinerant inebriate would have an infinite choice of directions. However, most simulations limit the directions and apply probabilities to each one.

With pixels selected as the dimensional unit each peripheral pixel is the same 'distance' away from the origin regardless of the size of the direction-encoding square, that is one 'step'.

4.10.2.2. Overlapping versus Non-overlapping Samples

Multiple bit generators which produce overlapping samples behave as a binary source with a memory, where the direction depends on the previous bits. For example, a two bit overlapping sample is the output of a single-bit generator with a memory of one bit.

Non-overlapping samples are the equivalent of a multiple symbol generator. For example, a four bit generator will produce 16 unique output sequences and these can be encoded as hexadecimal characters.

4.10.2.3. Square versus Round Targets

With a circular real-world target, the size of the step does not alter with the direction. Most simulations use a single step size and define the number of directions to permit the exercise to work. Using a very large number of directions with specific probabilities may make the solution of the problem very difficult.

In Pixel Space, it does not matter if the pixel is not square, as long as each step in a particular direction is always the same length in pixels. This can be visualised as a non-Euclidean space where the magnitude of a standard length varies depending on absolute direction and / or location. If all entities within this space comply with the same conditions then, subjectively, a unit step in any direction has the same length while to an observer outside of the space, objectively, it can be observed that step lengths differ depending on direction.

Figure 4.10 shows the periphery of the targets is the same number of steps from the origin in any permitted direction and consequently, within the definitions given, the square in pixel space behaves as a circle for the purposes of the proposed tests.

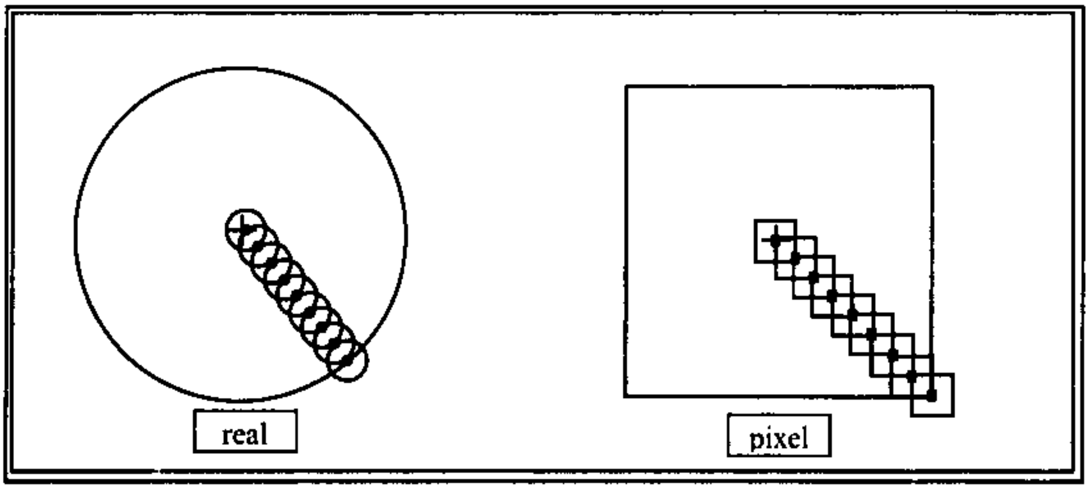


Figure 4-10 Comparison of Real and Pixel Based Walks

4.10.2.4. Errors in Representation.

A major concern is that the use of a circular target may introduce an error when a square target is necessary due to the underlying data structure of pixel squares.

Using a round target may underestimate the generator characteristics due to the target area being irregular. While the target looks like a circle to the observer in real space, using pixels as the basic unit means that it is not a circle.

In the real world, if the actual target is a square and a circle is inscribed inside the square then the relationship of the respective areas is given by the area of the circle divided by the area of the square.

$$\pi r^2 /(2r)^2 = \pi/4$$

Equation 4-13

Using equation 4.13, when the radius of the circle is 100 then the circle area is 31415.92 and the square area is 40000. The relationship between them is 0.78539 or approximately 79% which is equal to π /4 of the area of the square.

The circular target, in pixel space, is then smaller than required and the trace terminating at the periphery is actually terminating earlier than if a square target is used. Consequently, if the square target is the equivalent of a circle in pixel terms then the circle is not really circular in pixel terms.

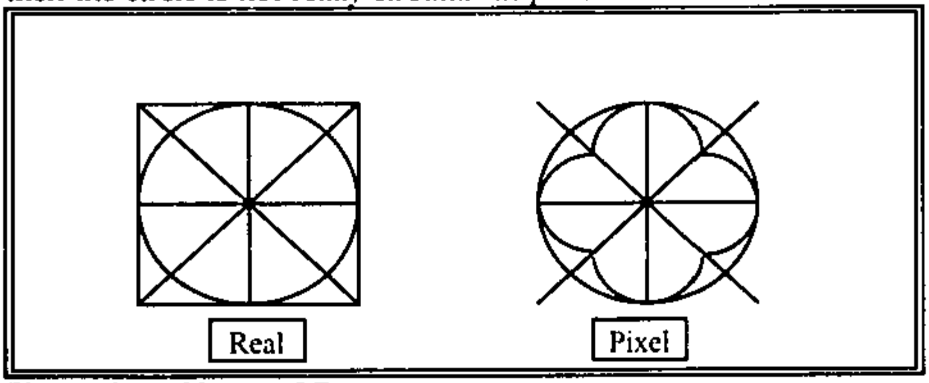


Figure 4-11 Shape of Target

The picture on the left expresses the relationship between the Square target and the circular target in the real world. The picture on the right expresses the circular target, the white area, as it would be seen in pixel space where the square actually is a circle in pixel space.

The relationship is between the two areas in pixel space is approximately 78%.

4.10.3. Summary

- Drunkard's Walk Simulations are easily implemented.
- Multibit representations of direction may require substantial resources and a very large test display area.
- Apart from some simple statistics generated evaluation of the results requires further research.
- Results are included in the Appendix 9-19.

4.11. Ordering Paradigm

4.11.1. Introduction

As testing a complex function with a large range of inputs may be infeasible it is important to develop a simpler test that will give some assurance that the function generator is not flawed. Comparison of the function input and output sequences is to provide some assessment of whether the function is one-to-one.

4.11.2. Methods Used

The concept of a randomness in a binary sequence implies that for each bit in the sequence, the probability of the next bit being either the same or different is equal, that is, there is no preference for one value or the other and no dependency on previous bits.

It is considered that comparison of the similar length input and output binary sequences of a function generator would show if any relationship exists between them.

Hamming 'weight' and 'distance' are concepts covered in Error-Detecting Codes where 'the Hamming Distance between two words is defined to be the number of positions in which the words differ' (Peterson,1960, p 7) and the 'Hamming Weight' of a sequences 'is defined to be the number of non-zero components' (Peterson,1960, P30) For example, for the words 11010 and 10001, their Hamming Weights are three and two respectively and the Hamming Distance between them is three.

It is proposed that the average Hamming Weight and Distance may be a useful indicators of the transfer function describing a particular process and specifically where an input sequence is processed into an output sequence according to some unknown function.

An arbitrary binary sequence of n bits is defined as $\underline{X}$, and X_i is an instance of $\underline{X}$. An operation f() can then be performed on X_i an yielding $f(X_i)$ or X'_i

$$X'_i = f(X_i)$$

Equation 4-14

For example, with X_i of 0101 processed by f() to obtain 1010 as the sequence X'_i . The Hamming Distance of these two sequences is 4 and the Hamming Weight of both the sequences is 2.

It is suggested that the Average Hamming Distance between input and output sequences may provide a measurable characteristic of f() when each unique value of x_i , that is from 0 to 2^n-1, is processed sequentially.

	Reference		F(x)Rotate			F(x)Left shift + 1		
	Input	Wt.	Output	Wt.	Distance	output	Wt.	Distance
1	000	0	000	0	0	001	1	1
2	001	1	010	1	2	011	2	2
3	010	1	100	1	2	101	2	3
4	011	2	110	2	2	111	3	1
5	100	1	001	1	2	001	1	2
6	101	2	011	2	2	011	2	2
7	110	2	101	2	2	101	2	2
8	111	3	111	3	0	111	3	0
	Total	12	Total	12	12	Total	16	13
	Avg.	1.5	Avg	1.5	1.5	Avg.	2	1.75

Table 4-9 Comparison of Input and Output Sequences

For example, using a complete set of 3 bit binary sequences in order from 000 to 111, that is from 0 to 2^3-1 or 7 and defining f() as a rotation without loss of data of one bit such that the sequence 010 becomes 100, then all other sequences of 3 bit length are processed.

In Table 4-9, comparing the bitwise similarities between input sequences and output sequences for complete range of inputs gives an average value for all 8 sequences of 1.5 which is half the number of bits. Additionally it is noted that the function f(x) is 'one-to-one' in that there is a unique output sequence for each unique input sequence.

Using a function that shifts the sequence left by one position and increments the result by one it is noted that the function is not 'one-to-one' and the average value of matches is 1.25 which may indicate that for functions that are not 'one-to-one' some change in the bitwise matches may be detected.

This characteristic may be of value in determining if a function is likely to be 'one-to-one' for functions of much greater complexity and length, especially when it is impractical to generate and test all possible unique sequences.

Considering a function generator that applies an algorithm such as the block cipher Data Encryption Standard (DES) which uses a 64 bit key to encrypt a multiple sequences of 64 bit blocks. It is noted that the number of unique input sequences and key sequences are both 2^{64} or 1.844 * 10^{19} and that generating and comparing all the unique sequences may be infeasible.

For an individual message, X*, with all possible unique key sequences, $\underline{K}$, to the encryption process, the output from the encryption process may be determined.

$$X'_i = f(X^* , K_i)$$

Equation 4-15

In an encryption the relationship between the input and the output must be 'one-to-one' and 'one-way', that is each unique input must produce a unique output and it must not be practical to determine the input from knowing the output.

4.11.2.1. Example of Average Hamming Distance

For sequences of n bits the Average Hamming Distance will be approximately equal to half the number of bits if the process f(x) is uniformly distributed.

$$AHD = n/2$$

Equation 4-16

To explain the concept further, Table1 shows two separate messages, X_1 = 1010 and X_2 = 1011, arbitrarily encrypted by function $\underline{K}$ where K_i is an instance of $\underline{K}$ and the resultant encryptions, K_iX_1 and K_iX_2 compared with the message for matching bits in each position.

In Table 4-10, the first sequence the average Hamming Distance *per bit* is 17/32 or 53.127% and in the second sequence that is 13/32 or 40.625%. It is considered that any substantial or persistent deviation from the 50% nominal value is indicative of some type of flaw in the process.

For example, an encryption, K_a , is defined such that each bit in the message, X_a , is inverted, then for all $\underline{X}$, the average Hamming Distance is n, the number of bits in the message X_a . Conversely, an encryption, K_b , such that no bit in the message, X_a , is ever inverted and no change is made to the message, means for all $\underline{X}$ the average Hamming Distance is 0.

	x_1 = 1010		Bits		x_2 = 1011		Bits	
i	Function	k_ix_1	HD_i	Wt.	Function	k_ix_2	HDi	Wt.
1	k_i	0101	4	2	k_i	1101	2	3
2	k_i	1101	3	3	k_i	1001	1	2
3	k_i	1111	2	4	k_i	0010	2	1
4	k_i	0000	2	0	k_i	1110	2	3
5	k_i	0111	3	3	k_i	0101	3	2
6	k_i	0011	2	2	k_i	1010	1	2
7	k_i	1010	0	2	k_i	1011	0	3
8	k_i	1011	1	3	k_i	0001	2	1
	No. Of bits	32	17	19	No. of bits	32	13	17
		Avg.	4.25	4.75		Avg	3.25	4.25

Table 4-10 Simulation of Ordering Paradigm

The long term behaviour of a particular encryption method may be examined with this method. In Table 4-10 the average values show that the function may not be one-to-one and further testing may be needed.

For example, after 1 * 10^6 iterations of an encryption method, K_c , on a message X_c , the Average Hamming Distance per bit may be of 0.65 and this characteristic may be significant to the user.

4.11.2.2. One to One Function

A simple example of a one-to-one function occurs when the bits of the input are transposed.

If a bit in the input goes to more than one bit in the output then it is no longer a one-to-one function. By summing the number of similarities between the input and output the average Hamming Distance can be calculated.

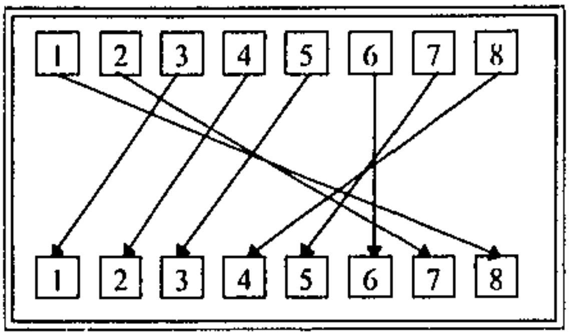


Figure 4-12 Transposition diagram for a One-to-one Function

For functions where the number of bits is much larger, for example, n = 1 * 10^6 the amount of work to tabulate the comparisons of outputs and inputs increases exponentially according to 2^n and it may not be feasible to generate all inputs and outputs for analysis. Consequently, some kind of sampling or testing regime needs to be developed. Figure 4.12 shows a diagrammatic representation of a one-to-one function.

4.12. Summary

4.12.1. Hardware Design

- The initial design is practical but will require substantial prototyping.

4.12.2. Analysis of a Transaction

- Electronic transactions present opportunities for fraud.
- The proposed implementation specifies a procedure to follow to minimise the opportunities for fraud.

4.12.3. Proposed Implementation of Function Generator

- The proposed method is simple to implement
- Limitations are the size of the randomiser and the amount of memory needed to manipulate the bits

4.12.4. Randomiser Construction

- Non-Deterministic Randomisers must be very large to be used in the implementation
- Deterministic Randomisers may not have sufficient length or complexity to provide security to the proposed method
- A hybrid method of constructing Deterministic Randomisers may address the length and complexity issues

4.12.5. Large Numbers in Calculations

- Calculation of Large Factorials and Exponentials is constrained by accuracy and speed. The more significant digits used the slower the calculation and the greater the precision.
- Expansions of large factorials and exponentials provides long sequences of integers for use within the implementation

4.12.6. Test T

- The Test-T is practical and easy to implement but needs to be validated before use.

4.12.7. Classification Scheme

- The classification scheme proposed provides a means of characterising random sequence generators by their outputs.
- The scheme may also provide means of specifying the combination of random sequence generators

4.12.8. Visual Testing

- Visual testing can be implemented easily and used to look for flaws in a wide range of random sequence generators.
- Target size and shape need to be considered in running the tests.
- Using Overlapping or Non-overlapping samples depends on the nature of the tests desired.

4.12.9. Ordering Paradigm

- Bitwise comparison of the input and output sequences of a function generator can be used to estimate function characteristics.
- The Average Hamming Weight and Average Hamming Distance of a process can be used to characterise a function.

5. User Identification Based On A One Way Function

5.1. Specification of Design Objectives

5.1.1. Introduction

It may be practical to legally assign a particular Smart Card to a person by recording the unique aspects of the card and the identity of the responsible person. Once this is achieved the person can use that card as the equivalent of their personal 'seal' or signature.

These signatures may be valid because the User and the card are positively identified and the User has to initiate the card at the correct part of the transaction cycle and record the results. The actuation of the card signifies the User's assent to the transaction. Each generated signature must be unique to guard against misuse yet still be attributable to the specific User to prevent fraud.

5.1.2. Method Used

This can be achieved by programming into the permanent memory of the Smart Card an array of random bits and a selection algorithm. For a long random sequence there is a very small probability of another sequence being close to it in content. Smart Cards using this concept can be unique even when many millions of them are created. Groups of cards with identical randomisers may differ in the selection algorithm or vice versa.

The number of unique randomisers available to fit the allocated memory space is very large. For example, a randomiser of 2000 bits is one of a set of 2^{2000} or $1.148 * 10^{602}$ possible randomisers. Smart Cards may contain up to 20 kilobytes or 163840 bits of permanent storage (Rankl & Effing, 1997, p430) which will contain the program and randomiser as well as any additional applications.

The Smart Card must store a copy of all input sequences with the output sequence generated so that the user can automatically keep the summary of the agreements for future reference.

Additional activities such as validation of agreements and secure communications with this method are desirable.

5.1.3. Summary

- Each Smart Card must be unique in some way to identify the card and must possess unique properties that can be used to provide a unique signature or validation of transactions.
- The Smart Card must be associated with an individual by means of a photograph and personal details or other method.
- The Smart Card must use some form of personal actuation such as a pin number or biometric identification.
- Sequences produced by the card must be unique and repeatable.
- A Smart Card must be physically secure against illicit operations to discover its contents.
- Additional functions may include hash and cryptographic algorithms.

5.2. Description of a Smart Card Design

5.2.1. Introduction

It is practical to specify the characteristics of a Smart Card to meet these requirements. The card must be physically secure and legally assignable to a person or company. It must implement the algorithm and store transaction details. Additionally, it must implement secure hash and data encryption methods and produce a serial number to identify the card uniquely.

The major requirement is memory rather than processing speed, as many hash values are 128-256 bits and may take less than a second to transfer even at Smart Card serial speeds. The memory must be divided physically into Read Only Memory (ROM) which cannot be determined without destroying the chip and Random Access Memory (RAM) to store the working data and summary information. It may be necessary to store the summary data in a 'write-once, read-many' (WORM) memory or Electrically Erasable Programmable Read Only Memory (EEPROM) so the card can act as a receipt by storing the input sequence used to stimulate it.

An encryption algorithm is used to encrypt digital data for storage and transmission.

A symmetric cipher requires that a copy of the key is available at both ends of the system. The sender and receiver must share this information which is as confidential as any message it is used to encrypt.

Figure 5-1 describes the proposed design for a Smart Card. The control functions are indicated by dotted lines; the data flows by solid lines.

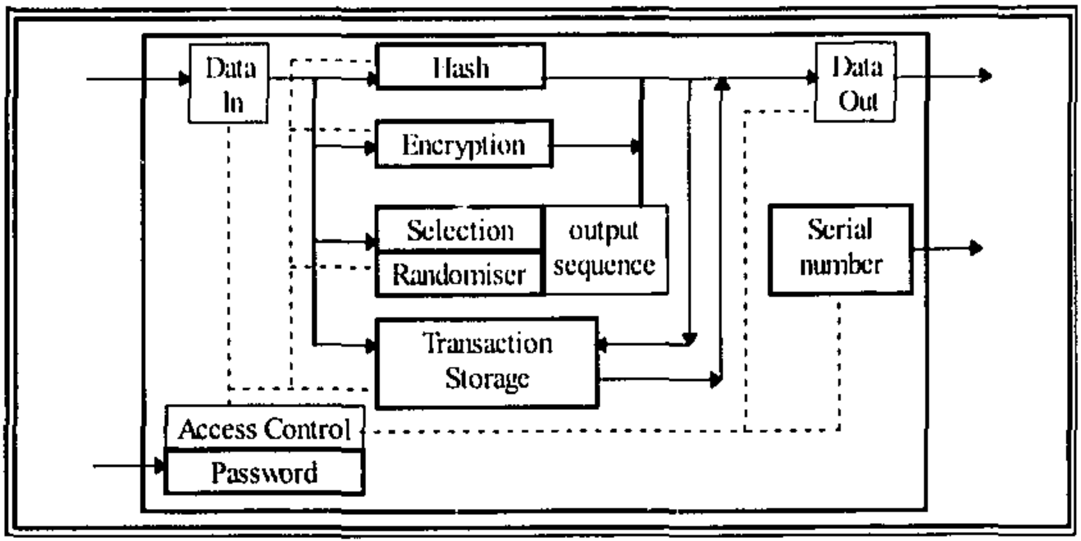


Figure 5-1 Schematic of Proposed Smart Card Design

The Smart Card input sequence may be used to generate a key for a Block Cipher such as the Data Encryption Standard (DES). As the output sequence is unique to each Smart Card, the input sequence can be transmitted as the message header with the encrypted message.

A 'Block Cipher' is an algorithmic method of processing fixed-length blocks from a message. (Schneier,1996, p270) The DES algorithm performs a substitution and shuffling of the bits according to a preset algorithm for 32 iterations and then selects and processes the next block. A typical block is 64 bits and a typical key could be as short as 64 bits or as large as 196 bits in a DES variant such as 'Triple-DES'.

A 'Stream Cipher' (Schneier,1996, p197) is an algorithmic method of processing a message one bit at a time by performing 'addition modulo-2' with the key sequence. The security of the Stream Cipher rests solely on the characteristics of the key sequence which must be at least as long as the message being encrypted.

The Smart Card output sequence can be prolonged to provide a key sequence and this implies the ability to produce sequences of varying lengths or the ability to use selected portions from the output sequence.

Where provision exists for Smart Cards to be used on public phone terminals it is considered possible but not immediately practical to encrypt messages with the Smart Card. Personally encrypted secure communications on public carriers may be implemented separately with the data on a lap-top or a digital voice message being encrypted before being transmitted. Limitations in the Smart Card data transfer rates may make real-time voice encryption difficult to implement.

For example, a standard sound file format on many PCs is the 'Wave' file identified by a '.WAV' suffix to the file name. Software is readily available to capture a sound sample and off-line processing is relatively straight forward. Sampling at the minimum rate of 11025 eight bit samples per second (monophonic) produces a file of 661500 bytes per minute of sound. Speech is able to be compressed substantially with proprietary compression software such as 'PKZIP'. Research shows that approximately 300 seconds (5 minutes) of speech can be compressed and stored on a single 1.44 Megabyte floppy disk. Off-line encryption of this compressed data file is readily achieved.

The card should produce a digital serial number which can be compared against the markings on the card. Additionally, a second serial number can be generated from the randomiser which will identify the randomiser on the card.

Access control to the card may use a password or biometric identification. Biometric identification is difficult at present to implement but advances in technology may produce fingerprint recognition or some other method that is able to be implemented on a Smart Card.

The card must be able to use at least one publicly available hash function or produce a hash value of an input sequence.

All transactions must be able to be securely stored and if necessary securely transmitted on public carriers.

Memory size and access speed are the major criteria in selecting a Smart Card to produce this application.

The current state of development of the Smart Card with the memory available and the low rate of data transfer may mean that some of these functions are impractical though changes in the available technology may alter this.

5.2.2. Summary

- ❑ A method of generating validation sequences or electronic signatures is necessary for some forms of electronic commerce.
- ❑ The method must minimise opportunities to commit fraud and provide security against misuse
- ❑ The proposed design addresses the following issues:
- ❖ Card identification
- ❖ Card User identification and secure access
- ❖ generation of hash, validation and signature sequences
- ❖ key distribution in a symmetric cipher system
- ❖ record keeping
- ❑ Implementation requires
- ❖ The users of the method must be able to confirm the results of the process at any time.
- ❖ The transaction must be recorded and any changes must be detectable.
- ❖ Restriction of the opportunities for misuse
- ❑ A practical specification for a Smart Card design incorporating multiple functions depends on the software simulation to provide information for a prototype hardware implementation.
- ❑ Implementation of some functions may be impractical due to
- ❑ Development time and capabilities of the Smart Card

5.3. Description of the Theoretical Method

5.3.1. Introduction

The theoretical method 'A Provably Secure, Strongly-Randomised Cipher' (Maurer,1990, p361) proposes 'provable security .... based on the availability of a very large publicly accessible string of random bits'. The criteria for provable security are that the randomiser is substantially larger than all the plain text encrypted with it and that the key information is the starting point within the array of randomised strings blocks used to produce the key string.

Maurer concludes with two recommendations and a conjecture. One recommendation relates to developing the randomiser in time, requiring an opponent to process a substantial part of the randomiser to be reasonably certain that the decryption of the message is correct. The second recommendation is that the selection of bits from the randomiser is made more difficult thereby permitting the shortening of the Randomiser and rendering an application more feasible. Furthermore, Maurer states:

"Finally, we would like to point out that randomization techniques similar to those presented in this paper may be useful for the construction of practical ciphers, even when the randomizer is not sufficiently long to guarantee a reasonable lower bound on the enemy's computational effort required to break the cipher or when the randomizer is replaced by a pseudo-random sequence" (Maurer,1990, p372)

5.3.2. Description of the Theoretical Method

This section makes substantial reference to the original paper by Maurer. (Maurer,1990)

Random variables are denoted by capital letters, whereas the corresponding small letters denote values taken on by these random variables.
A binary additive stream cipher
$\underline{R}$ = publicly accessible binary random string consisting of K blocks of length T and total length of L= KT bits. The blocks are denoted by R[k,0],...,R[k,T-1] for $1 \le k \le K$. (length approximately 10^{20}) See Table 6-1.
$\underline{W}$ = $[W_1 , \ldots , W_N]$ binary Keystream of length N
$\underline{X}$ = $[X_1 , \ldots , X_N]$ binary Plain text of length N
$\underline{Y}$ = $[Y_1 , \ldots , Y_N]$ binary Cryptogram of length N
$\underline{Z}$ = $[Z_1 , \ldots , Z_K]$ a secret key where $Z_k \in \{0, \ldots , T-1\}$ for $1 \le k \le K$ specifies a position within each block of $\underline{R}$. Chosen to be uniformly distributed over the key space $S_{\underline{Z}} = \{0, \ldots ,T-1\}^K$ The number of bits to represent the key is K $\log_2$T.

The cryptogram $\underline{Y}$ is obtained by adding $\underline{X}$ and $\underline{W}$ bitwise modulo 2:

$$Y_n = X_n \oplus W_n \quad \text{for } 1 \le n \le N$$

Equation 5-1

R[1,0]	R[1,1]	...	R[1,T-1]
R[2,0]	R[2,1]	...	R[2,T-1]
...	...	...	...
...	...	...	...
R[K,0]	R[K,1]	...	R[K,T-1]

Table 5-1 Description of Randomizer $\underline{R}$ as a two dimensional array of Length = K * T

Keystream $\underline{W}$, a function of key $\underline{Z}$ and randomiser $\underline{R}$ is the bitwise modulo 2 sum of K subsequences of length N within the randomizer starting at positions specified by the secret key. Each block (row) of R is extended cyclically, that is, each row is reduced modulo T:

$$W_N = \sum_{k=1}^{K} \oplus R[k,(n - 1 + Z_K)\bmod T]$$

Equation 5-2

for $1 \le n \le N$, where $\sum\oplus$ denotes summation modulo 2.

Z1	R[1,Z1+0]	R[1,Z1+1]	...	R[1,Z1+N-1]
Z2	R[2,Z2+0]	R[2,Z2+1]	...	R[2,Z2+N-1]
...	...	...	...	...
...	...	...	...	...
ZK	R[K,ZK+0]	R[K,ZK+1]	...	R[K,ZK+N-1]
$\sum\oplus$	W0	W1	...	WN-1
Plain	X0	X1	...	XN-1
Y=W⊕X	Y0	Y1	...	YN-1

Table 5-2 Description of Cryptogram $\underline{Y} = \underline{W} \oplus \underline{X}$ from Randomizer $\underline{R}$ with Secret Key $\underline{Z}$

The keystream, W, described by equation 5.2, is the bitwise modulo 2 sum of K sub-sequences of R which are of length N, that is, W1 is the sum modulo 2 of R[1,Z1] , R[2,Z2] .... , R[K,ZK] where ΣΦ indicates sum modulo 2. (See Table 5-2)

For example, where K = 50 and T = 10^{40} and N = 2^{30} (approximately $1*10^{9}$) gives a keysize of only 3320 bits, that is 50 $\log_2 10^{40}$. The legitimate user must only examine 50 randomizer bits per plain text bit while an enemy has to examine, even with an optimal strategy, perhaps 1/4 of all the bits per plain text bit. This gives M = KT/4 = $1.25*10^{41}$ bits in total or 1.16 * 10^{12} bits per plaintext bit. The chance of gaining more information about the plaintext is not greater than 10^{40} * $(1/4)^{50}$ which is less than 10 ^{21} .(Maurer,1990, p368)

5.3.3. Implementation of Theoretical Method

A trial implementation with a randomiser of 1 * 10^{6} bits demonstrates the ease of the theoretical method but ignores the requirement for a randomiser of much greater size. A method of meeting this requirement with deterministic construction is discussed later.

The arrangement of the randomisers in an array of K sub sequences of length T is easy to simulate in software. The secret key, Z, indicates the start of the sequence of bits selected from each block of the randomiser.

The index within each block, reduced modulo T, is shown by a dotted line in each block which permits the randomiser block to be viewed as a circular buffer. If the blocks are rotated to line up each bit indexed by the secret key Z then the bits in each block can be read vertically and summed modulo 2 to generate key string W.

The implementation divides a randomiser file into blocks of equal length and then fills a two dimensional array with contiguous bits from each block. Each column within the array is summed modulo 2 to produce a key bit which is used to encrypt a message bit. The security of the method relies on the qualities of the randomiser and the secret key. The use of numerous randomiser blocks to provide multiple bits for the creation of a key bit serves to hide the structure of the randomiser from analysis.

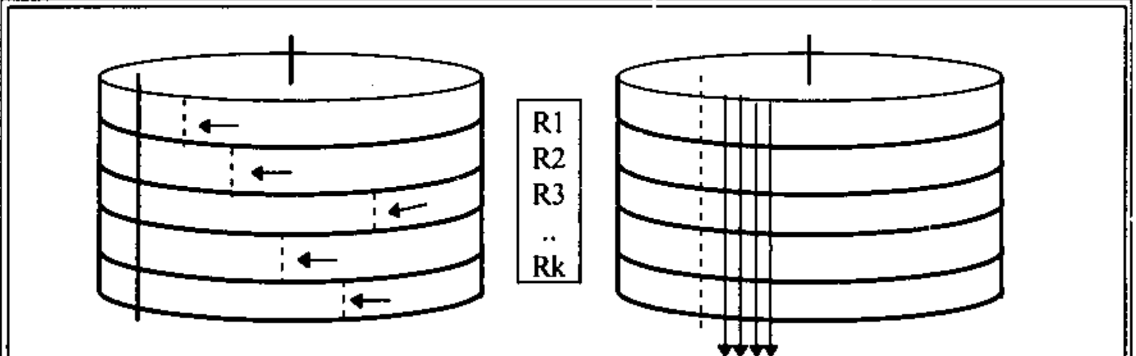


Figure 5-2 Schematic of Theoretical Randomiser Construction Method

Repeating Maurer's analysis, where K = 10 and T = 10^{5} and N = 2^{10} (approximately 1024 bits) gives a keysize of approximately 166 bits, that is 10 $\log_2 10^{5}$. The legitimate user must only examine 10 randomizer bits per plain text bit while an enemy has to examine, even with an optimal strategy, perhaps 1/4 of all the bits per plain text bit. This gives M = KT/4 = 2.5 $*10^{5}$ bits in total or 244 bits per plaintext bit. The chance of gaining more information about the plaintext is not greater than 10^{5} * $(1/4)^{10}$ which is less than 10^{-2} .

5.3.4. Summary

- Implementation with a small randomiser is practical but the ease of implementation is offset by the loss of security.

- A small randomiser compromises the security of the method by reducing the analytical workload and increasing the likelihood of randomiser bits being reused.
- The effective working life of the cipher would depend on how much information about the randomiser is released with each decrypted message.
- A much larger randomiser is constrained by the amount of time and resources needed to store and search the array for the relevant bits. Consequently a trade-off between size and practicality is needed.

5.4. Description of the Proposed Method

5.4.1. Introduction

The theoretical method is limited in implementation by the size of the randomiser required and the key information is proportional to the number and size of the blocks used. Speed and memory constraints may make the implementation impractical.

The proposed method is developed from original research for an undergraduate project where a circular buffer of binary data of length $5 * 10^9$ bits is traversed and bits are selected or discarded according to a simple algorithm. The algorithm needs to specify a Start Point in the Buffer and a method of selecting the necessary number of bits.

The benefits of the proposal include use of a smaller randomiser, key size is not directly related to the size of the randomiser and the speed is comparable to Maurer's method.

5.4.2. Description of Proposed Method

A random binary sequence $\underline{R} = [R_{0,} R_1, \dots , R_{N-1}]$ is generated and stored in a randomly accessible medium. A constraint of $\underline{R}$ is that it is substantially larger than any message it is used to encrypt, that is N >> n. It may help in understanding the proposal by imagining the randomiser as a circular buffer able to be entered at any point and traversed in either direction.

Given a binary message, $\underline{M} = [M_{0,} M_1, \dots , M_{n-2,} M_{n-1}]$, a key, $\underline{K} = [K_{0,} K_{1,} \dots , K_{n-2,} K_{n-1}]$, the encrypted binary message, $\underline{E} = [E_{0,} E_1, \dots , E_{n-2,} E_{n-1}]$, is generated by bitwise addition modulo 2 of $\underline{M}$ and $\underline{K}$. (where $\oplus$ is addition modulo 2).

$$E_n = M_n \oplus K_n \text{ for } 1 \leq n \leq N$$

Equation 5-3

To produce the binary keystream, K, the indices of the selected bits in the Randomiser, $\underline{R}$, are specified by the Index Sequence, $\underline{I} = [I_{0,} I_1, \dots , I_{n-2,} I_{n-1}]$.

If an arbitrary integer starting point in R denoted by SP and a sequence of integers $\underline{S} = [S_0 ,S_1, \dots , S_{j-2,} S_{j-1}]$ are defined then the Index Sequence, $\underline{I}$, to Randomiser, $\underline{R}$, can be generated where $I_0 = SP$, $I_1 = SP + s_0$, … ,$I_N = SP + s_0 \dots + s_x$. The values in S are either part of the absolute index of a bit in R or indicate the number of bits to discard between bits which are selected.

The key generation algorithm is defined as

$$K_i = R_{d \bmod N}$$

Equation 5-4

where d is the index of the bit in $\underline{R}$ reduced modulo N, that is $I_x = d$:

1. Absolute Mode

$$d = SP + \sum_{i=1}^{n} S_{(i-1 \bmod j)}$$

Equation 5-5

or

2. Discard Mode

$$d = SP + \sum_{i=1}^{n} S_{(i-1 \bmod j)+i}$$

Equation 5-6

The selection mechanisms employed in this method are easily implemented and it is submitted that they meet the requirement of making the bit selection method more difficult and so permit a shorter randomiser to be used. (Maurer,1990, p372)

5.4.3. Implementation Example of Proposed Method

5.4.3.1. Definitions

Random binary sequence	$\underline{R}$	=	$[R_0, R_1, \dots, R_{N-1}]$
Plain binary message	$\underline{M}$	=	$[M_0, M_1, \dots, M_{n-1}]$
Encrypted binary message	$\underline{E}$	=	$[E_0, E_1, \dots, E_{n-1}]$
Key binary sequence	$\underline{K}$	=	$[K_0, K_1, \dots, K_{n-1}]$
Integer Index sequence	$\underline{I}$	=	$[I_0, I_1, \dots, I_{n-1}]$
Integer Selection sequence	$\underline{S}$	=	$[S_0, S_1, \dots, S_{j-1}]$
Integer Starting Point	SP		

where N is the size of the randomiser $\underline{R}$, n is the size of the message $\underline{M}$ to be processed and N >> n . An arbitrary integer, J, indicates the number of values used in the selection set $\underline{S}$. SP and S_x are constrained modulo N.

To produce the binary Key Sequence $\underline{K} = [K_0, K_1, \dots, K_{n-2}, K_{n-1}]$ from the Randomiser, $\underline{R}$, the indices of the bits to be selected are specified. This is accomplished by using the Index Sequence, $\underline{I} = [I_0, I_1, \dots, I_{n-1}]$.

An arbitrary starting point in $\underline{R}$ is defined by the integer SP and a sequence of integers $\underline{S} = [S_0, S_1, \dots, S_{j-1}]$ can then be used to generate the Index Sequence, $\underline{I}$, where $I_0 = SP$, $I_1 = SP + S_0$, $I_2 = SP + S_0 + S_1$, ... $I_n = SP + S_0 + \dots + S_X$.

The index for each bit in R, denoted by I_i, is defined as $I_i = d \bmod N$ where

1. Absolute

The selection method where the absolute index of the bit in R is specified. (Using equation 5.5)

$$d = SP + \sum_{i=1}^{n-1} S_{(i-1 \bmod j)}$$

Given N = 60, n = 6, j = 3, SP = 10, $\underline{S}$= [5, 7, 2] then
$\underline{I}$ = [SP, SP + S_0, SP + S_0 + S_1 , SP + S_0 + S_1 + S_2 , SP + S_0 + S_1 + S_2 + S_0 , SP + S_0 + S_1 + S_2 + S_0 + S_1] = [10, 15, 22, 24, 29, 36]
and
$\underline{K} = [RI_0, R\,I_1, R\,I_2, R\,I_3, R\,I_4, R\,I_5] = [R_{10}, R_{15}, R_{22}, R_{24}, R_{29}, R_{36}]$.

As an example: (hyphens inserted for readability, selected bits underlined)
$\underline{R}$ = {101001010$\underline{1}$-0100$\underline{1}$10101-0$\underline{0}$1$\underline{0}$10110$\underline{0}$-1011$\underline{1}$10101-1010100110-1010101001}
$\underline{K}$ = {110001}

2. Discard

The selection method where the relative index of the bit in $\underline{R}$ is specified, that is the number of bits to be discarded between selections. (Using equation 5.6)

$$d = SP + \sum^{n-1} S_{(i-1 \bmod j)+1}$$

i=1

Given N = 60, n = 6, j = 3, SP = 10, $\underline{S}$ = [5, 7, 2] then $\underline{I}$ = [SP, SP + S_0 + 1, SP + S_0 + 1 + S_1 + 1, SP + S_0 + 1 + S_1 + 1 + S_2 + 1, SP + S_0 + 1 + S_1 + 1 + S_2 + 1 + S_0 + 1, SP + S_0 + 1 + S_1 + 1 + S_2 + 1 + S_0 + 1 + S_1 + 1]
$\underline{I}$ = [10, 16, 24, 27, 33, 41] and $\underline{K}$ = [R_{10}, R_{16}, R_{24}, R_{27}, R_{33}, R_{41}]

As an example: (hyphens inserted for readability and selected bits underlined)
$\underline{R} = [101001010\underline{1}\text{-}0100\underline{1}0101\text{-}001\underline{0}10\underline{1}100\text{-}10\underline{1}110101\text{-}\underline{1}010100110\text{-}1010101001]$
$\underline{K}$ = [110111]

It can be seen that the same selection regime can produce two different Key sequences depending on the mode selected.

If the Randomiser is considered to be generated by a non-deterministic or random source then any reasonable sample of the Randomiser is itself random. An unreasonable sample would be choosing bits to meet a predefined objective such as selecting 1's half as frequently as 0's. This biased method is generally not suitable for statistical sampling.

It may help in understanding this process if it is considered to be another sampling regime to be compared with the traditional cluster, strata and other types of statistical sampling, i.e. a pseudo-random sampling.

5.4.4. Summary

- The proposed method is easy to implement and uses only one bit to encrypt each plain text bit.
- Key size is constant regardless of the size of the randomiser. The smaller randomiser is easily stored and traversed.
- The security of the proposed method depends on the qualities of the randomiser and the minimisation of the reuse of the bits. This can be achieved by limiting the uses and by varying the startpoint and selection algorithms.

5.5. Simulation of Theoretical Method Using Proposed Method

5.5.1. Introduction

The theoretical method can be implemented by the proposed method with the simple modification of choosing a sequence of bits and summing them modulo 2 to generate a single key bit. The secret key $\underline{Z}$ in the theoretical method is a set of integers used as Start Points. The number of bits used to generate a key bit, V, and $\underline{S}$ = [1] are used to select consecutive bits in absolute mode.

Each key bit in $\underline{K}$ = [K_0, , K_1, ... , K_{n-2}, K_{n-1}] is the sum modulo 2 of V bits in $\underline{R}$:

$$K_i = \sum_{v=0}^{V-1} \oplus R[Z_v + i]$$

Equation 5-7

5.5.2. Description

The simulation in the Absolute Mode entails a set of Start Points, $\underline{H}$ = [SP_0 , SP_1 , ... , SP_k] equivalent to the vector $\underline{Z}$ and Selection Set, S, = {1}. However, if the K segments of the randomiser are connected linearly into a ring buffer of length KL the indices of the chosen bits are determined with the proposed Method. See Table 5-3.

The theoretical method is considered to be a simple implementation of the proposed method which meets the requirements of a stream cipher application. Proof of security does not necessarily follow from Maurer's paper as the proposed method encapsulates the theoretical method.

SP_0	R[1,SP_0+0]	R[1,SP_0 +1]	...	R[1,SP_0 +N-1]
SP_1	R[2,SP_1+0]	R[2,SP_1 +1]	...	R[2,SP_1 +N-1]
...	...	...	...	...
...	...	...	...	...
SP_K	R[K,SP_K +0]	R[K,SP_{K-1} +1]	...	R[K,S_K + N-1]
Σ⊕	W1	W2	...	WN
plain	X1	X2	...	XN
W⊕X	Y1	Y2	...	YN

Table 5-3 Specifying Start Points

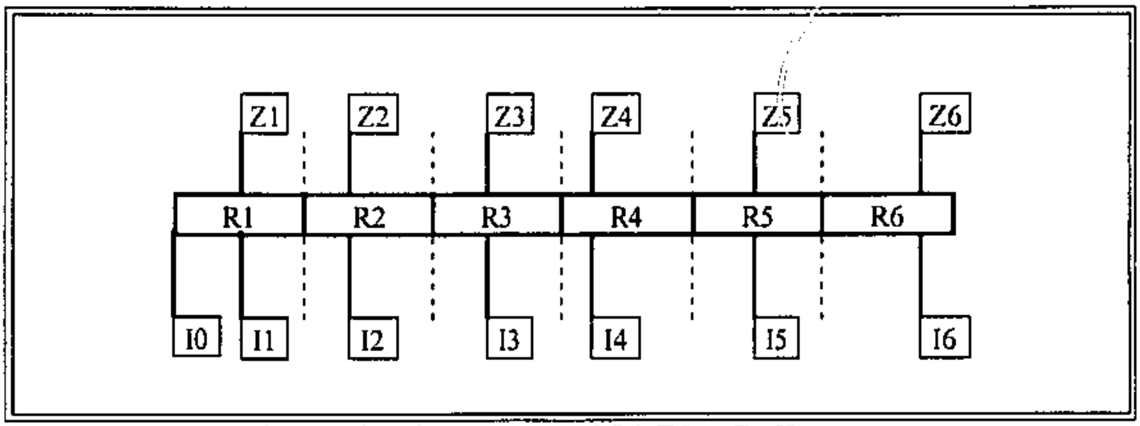


Figure 5-3 Maurer's Randomiser As Part Of A Ring Buffer

In figure 5.3, theoretical randomiser segments are concatenated to produce a ring buffer. The Z flags show the bit specified by the Key Z in each Randomiser segment. The I flags show that each value in Z can be described by an index calculated from the start of R1.

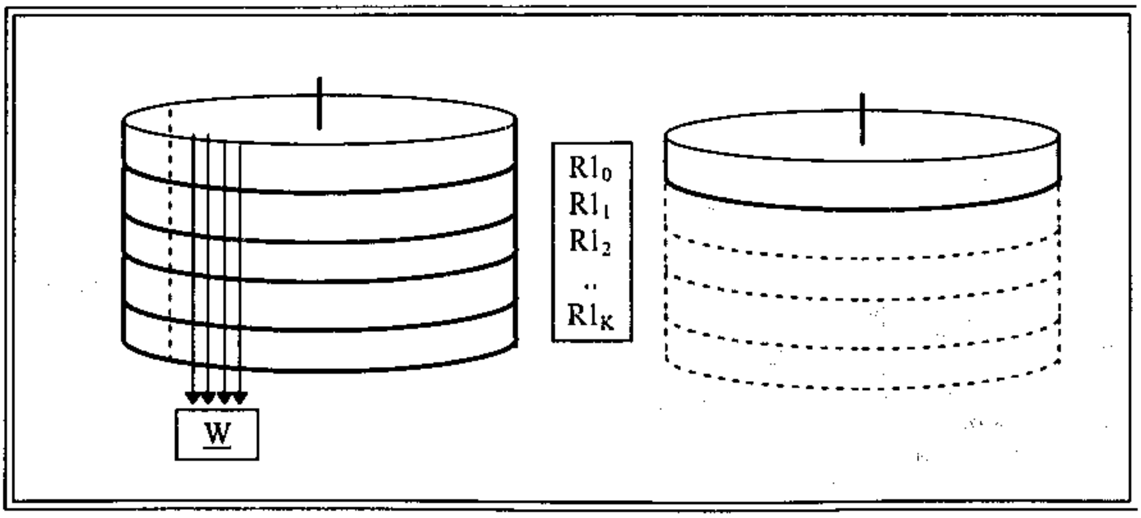
Table 5.4 shows that each Index flag can be calculated from the start of the R̲ and S shows Start Point and Selection Set in the Absolute Method.

	1	2	3	4	5	6
Randomise r	R1	R2	R3	R4	R5	R6
Key	Z1	Z2	Z3	Z4	Z5	Z6
I =	Z1	R1+Z2	R1+R2+Z3	R1+R2+R3+ Z4	R1+R2+R3+ R4++Z5	R1+R2+R3+ R4+R5+Z6
S =	SP	SP+S1	SP+S1+S2	SP+S1+S2+ S3	SP+S1+S2+ S3+S4	SP+S1+S2+ S3+S4+S5

Table 5-4 Calculation of Indices

Figure 5-4 Schematic of Randomiser Construction

Each block of R̲ is numbered in sequence from 1 to K, that is, R̲ = [R1 + R2 + , ... , RK]



In figure 5.4, the same randomiser block is copied and concatenated to produce the randomiser. The rotation of each block by the secret key value aligns the repeated blocks to permit bit selection. This method is equivalent to using each secret key value as a start point in the randomiser.

5.5.3. Summary

- ❑ The proposed method can easily reproduce the theoretical method and provides more opportunity for varying the key generation algorithm. However, the selection of contiguous bits is considered to be detrimental to the security of the method and should be avoided where practical.
- ❑ While the User may encrypt a message many times using different Selection Algorithms and sum them modulo 2 there may be little benefit in this.

5.6. Commentary on Security of the Proposed Method

The only provably secure method in general use is the One-Time-Pad, (Schneier P15) which whilst secure needs to transmit as much randomiser as the message to enable a message to be processed. The randomiser must be random and secret, that is, known only to the authorised users. The main criterion is that the randomising data is only used once ever.

Maurer addresses the secure randomiser by specifying a large publicly available randomiser that provides the basis for part of the security of the method. The main consideration is that using multiple bits to encrypt each message bit obscures the actual structure of the randomiser and makes the reconstruction of the key generation method from a known message much harder.

The proposed method is provably secure, if and only if the selected randomiser bits are used once. Multiple use of the randomiser makes the method conditionally secure and the main mechanism for ensuring security is to limit the number of uses per randomiser. The use of different randomisers, selection sets and start points add to the method's security as it is practical to vary the components easily.

The key size of the Theoretical method is proportional to the number and size of the blocks. The key size of the Proposed method is constant and not proportional to the size of the randomiser.

5.6.1. Implementation on a Smart Card

Implementation on a Smart Card is practical and the limited use of a small randomiser with a complex selection method permits conditionally secure usage. The card can be changed rapidly if it is considered to be compromised or life-expired. The costs of implementation may be reduced if a standard card can be configured by the user. Randomisers on cards can be produced and publicly distributed so the users need to share Selection Sets and Start Points.

5.7. Smart Card Fraud

If a Smart Card is used repeatedly then it may be possible to collate all the outputs for analysis to reconstruct the algorithms and randomiser. Fraud or unauthorised access may not be easily limited if the reconstruction compromises the system. A substantial break in the system may not be detected until after a major fraud has been discovered. The loss of business reputation and the costs of repairing the damage may make the continued viability of the system impractical.

Estimations of the costs of fraud are proportional to both the number of cards in circulation and the number of uses of each card and a simple model is proposed to explain this.

The model uses estimations of the costs of the Smart Card and the costs of the implementation.

5.7.1. Fraud and Costing Model.

The estimation starts with a Hardware Cost which covers all the hardware needed to set up a system of readers and cards The Hardware Cost includes Card Costs, Reader Costs, Installation and Support costs.

Card Costs include a discount for quantity. The price per card reduces to 50% of the nominal card cost for bulk purchases.
Nominal Card Cost = $5.00
Card Cost = Nominal Cost * 0.5 + (Nominal Cost * 0.5)/sqrt(number of cards)
Total Card Cost = Card Cost * Number of Cards

Equipment Costs cover the number of readers needed and include a discount for quantity. The price per reader reduces to 50% of the nominal cost for bulk purchases.
Nominal Reader Cost = $50
Number of Readers = roundup (sqrt(number of cards / 3)
Reader Costs = Nominal Cost * 0.5 + (Nominal Cost * 0.5/sqrt(number of readers)
Equipment Costs = Reader Costs * Number of Readers

Installation Costs include the costs of cabling and accessories.
Nominal installation Cost = $5.00
Installation Costs = Number of Readers * Nominal Cost + 0.10 * (Number of Cards * Number of Readers)

Support Costs include ongoing costs of supporting a system.
Nominal Support Cost = $0.10
Support Cost = Number of Readers * Nominal Cost + sqrt(number of cards * number of readers)

Total Hardware Cost includes all costs in setting up and running the system.
Total Hardware Cost = Card Cost + Equipment Cost + Installation Cost + Support Cost

Hardware Cost Per Card = Total Hardware Cost / Number of Cards.

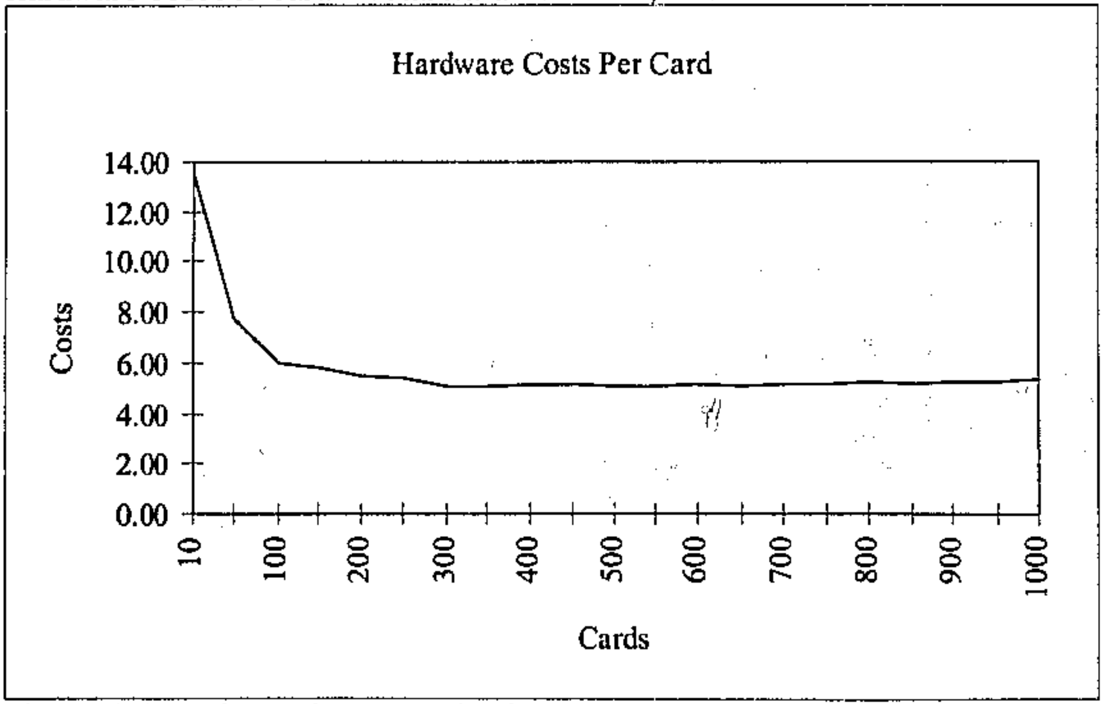


Figure 5-5 Hardware Costs Per Card

Using this model, the initial setup of the system is expensive but the costs per card drop quickly.

5.7.2. The Cost Per Use for a Smart Card

Nominal Cost Per Use = 0.05 + (0.05/sqrt(number of uses))

The Cost Per Card Per Use includes all the costs associated with multiple cards in regular use.

Cost Per Card Per Use = (Hardware Cost Per Card/number of uses) + Cost Per Use + 0.01 * sqrt(Sqrt(number of Cards * number of Uses))

The usage costs increase with the number of uses and the number of cards in circulation.

Fraud Cost Per Card per Use is proportional to the number of cards and the number of uses.

Nominal Fraud Cost of 0.90
Fraud Cost per Card Per Use = nominal cost * sqrt(sqrt(uses) * sqrt(cards))

These equations estimates are graphed in figure 6-6 to estimate the fraud costs per card per use.

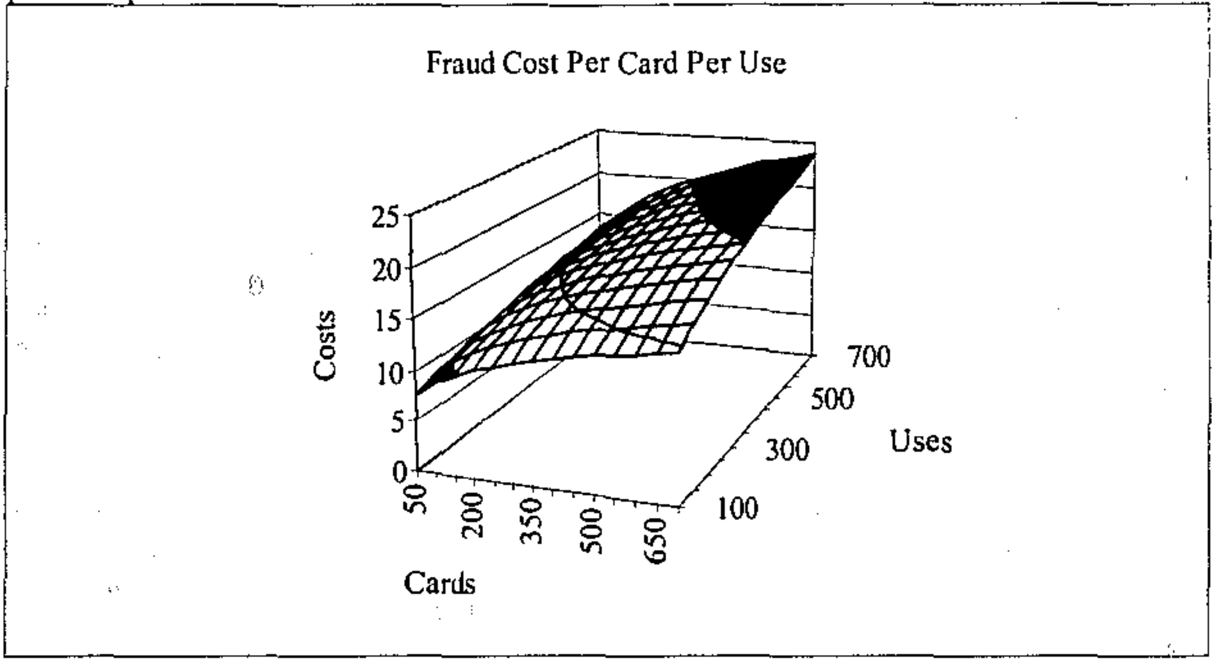


Figure 5-6 Fraud Cost Per Card Per Use

The costs of fraud increase with the number of cards in circulation, which permits an adversary to learn more about a system and also with the increase in uses which permit greater opportunities for fraud or errors.

Fraud Costs for a small number of cards used only once rise steeply as the possibility of fraud using inside information is likely.

With a large number of cards in circulation and many uses per card, the number of people who may commit fraud increases proportionally. Off-line transactions would be susceptible to small-scale frauds due to the lack of validation during the transaction.

5.7.3. Summary

- □ Using the estimations of the model approximately 500 uses per card is acceptable in keeping the costs of fraud per card use down. For example, given a transaction limit of $500 per day, it would take only one fraudulent transaction to give an average fraud of $1.00 per use on one card.
- □ A card may be used infrequently and 500 uses is considered to give about two years of use to the private user. This is estimated at once per working day, which is 5 * 48 * 2 or 480 uses.

5.8. Construction of Randomisers.

5.8.1. Introduction

A limiting condition of the Theoretical Method is the substantial length of the randomiser needed. A small modification is proposed which addresses this issue and hence allows a practical implementation of a large randomiser.

The size of the readily accessible randomiser is of the order of 1 * 10^{22} bits (1.25 * 10^{12} Gigabytes) and the constraints on storing and searching this sequence are substantial. However, a large randomiser can be produced deterministically from a smaller sequence and under certain conditions it may be indistinguishable from bits selected from a sequence produced by a random process.

If the randomiser is developed over time, that is broadcast publicly, the legitimate user need only listen when the necessary component is being broadcast while the unlawful user must copy and process all the randomiser incurring a substantial penalty in time and space terms.

For example, the randomiser could be widely distributed on a sequential set of Compact Discs and the legitimate user could collect only the necessary portions. However, the unlawful user must collect all the disks and be prepared to process all of them to decrypt the message. For example, CD's are commonly used to distribute software with magazines. Additionally, random sequences are also available. (Peterson,1998, p179)

5.8.2. Deterministic Construction of Randomisers

An operation, ROT(u), rotates a binary sequence of n bits by u bits without loss of data. For example ROT(1) on '101' gives '011', ROT(-1) on '011' gives '101' and ROT(3) on '101' gives '101' which is identical to ROT(0). If x is an integer rotation value then u = x mod n and n unique sequences can be generated by rotating the Basic Sequence by u from 0 to n-1.

A Complete Ordered Composite Sequence of n^2 bits can be produced by concatenating the n rotated sequences in order from 0 to n-1. However, concatenation can be performed in n factorial ways giving many possible Complete Composite Sequences.

For example, an 4 bit Basic Sequence produces four different rotated sequences and concatenation produces a 16 bit Complete Composite Sequence as one of 4 factorial or 24 possible 16 bit sequences. This process can be repeated with the 24 possible Complete Composite Sequences being concatenated, giving 24 factorial or 6.2 * 10 ^{23} possible sequences of 384 bits and so yielding a sequence of any desired length.

It is noted that the ROT() operation preserves some of the characteristics of the initial sequence. If P(0) denotes the probability of obtaining a zero for any bit in the Basic Sequence then ROT() does not alter P(0) in any of the rotated sequences, consequently, a Complete Composite Sequence or sequence comprising complete rotated sequences retains this characteristic.

A DeBruijn sequence has the characteristic of containing the complete set of overlapping binary sequences for some integer value. A simple example of a De Bruijn sequence of n = 3 is '00011101' (Johnsonbaugh, 1984, p127) which contains the set of overlapping three bit sequences '000', '001', '011', '111', '110', '101', '010', '100' that are a complete binary representation from 0 to 2^3-1.

It is noted that performing ROT() on the DeBruijn sequence does not alter this characteristic and the De Bruijn sequence can be considered as a circular bit buffer. Use of a De Bruijn Sequence for large n may be limited by the costs of generation and storage.

For example, using the eight bit DeBruijn sequence '00011101', the operation ROT(i) for values 0 to 7 can be performed and concatenated sequentially to get a Complete Ordered Composite Sequence with ordering = [0,1,2,3,4,5,6,7] (hyphens are inserted for ease of reading) :
000*111*01-00*111*010-0*111*0100-*111*01000-11010001-10100011-01000*111*-1000*111*0

Obvious subsequences such as '000' and '*111*' occur frequently though this may not be readily apparent in some of the possible factorial orderings. For example, a Complete Composite Sequence with Ordering = [6,3,0,1,2,4,5,7]
01000*111*-*111*01000-000*111*01-00*111*010-01110100-11010001-10100011-1000*111*0

Given a Complete Composite Sequence, ℵ, generated by the rotating and concatenating the Basic DeBruijn sequence, a sub-sequence ℵ' is extracted containing at least 5 complete Basic Sequences. Extraction of a smaller sub-sequences ℵ" and ℵ'" from ℵ is made to demonstrate the difficulty in determining the Basic Sequence with a reduced sample. (Note the hyphens '-' indicate boundaries of rotated Basic Sequences for ease of understanding)

ℵ = 01000*111*-*111*01000-000*111*01-00*111*010-01110100-11010001-10100011-1000*111*0
ℵ' = *111*-*111*01000-000*111*01-00*111*010-01110100-11010001-101
ℵ" = 1000-000*111*01-00*11*
ℵ'" = 0 or ℵ'" = 01

There is not enough information in ℵ" to readily determine the Basic Sequence and a number of short, separate subsequences such as ℵ'" provide little information about the Basic Sequence.

While structural similarities in the Composite Sequence exist due to the well ordered nature of the Basic Sequence, a random Basic Sequence with a randomly ordered set of rotations may make it more difficult to determine the Basic Sequence and ordering.

For example, a random integer, D, is converted into binary notation to get a Basic Sequence, B of length n. By choosing a random sequence of Rotations, ℜ, a Composite Sequence, ℵ, can be constructed by rotating and concatenating B by ℜ.

Given D = 179, B = 10110011, n = 8 and ℜ = [04618975] where $ℜ_i$ is constrained modulo n, a Composite Sequence, ℵ, is generated such that : (hyphens used for ease of reading)

For reference, a Complete Ordered Composite Sequence using ℜ = [0,1,2,3,4,5,6,7]
10110011-01100111-11001110-10011101-00111011-01110110-11101100-11011001

then a Composite Sequence ℵ using ordering ℜ = [0,4,6,1,0,1,7,5]

ℵ = 10110011-00111011-11101100-01100111-10110011-01100111-11011001-01110110
ℵ = 10110011-00111011-11101100-01100111-10110011-01100111-11011001-01110110
ℵ * = 111011-11101100-01100111-1011 ℵ * = 11101111101100011001111011

The sub-sequence '101' appears 8 times, the sub-sequence '1001' appears 6 times and determining if any binary sequence is composite may involve more than a cursory inspection. Determination of the Basic and Rotation Sequences to describe the incomplete Composite Sequence, ℵ * may be difficult.

If a substantial but undetermined number of small sequences are selected consecutively there may be enough data to identify the Basic Sequence (at least n bits and preferably much more) though determining the rotation details of the Composite Sequence is still difficult. Use of single widely separated bits may make it difficult to obtain any information about the Basic Sequence and it may be a non-trivial exercise to reconstruct the Composite Sequence.

5.8.3. Testing to Determine a Basic Sequence

To determine the length of the Basic Sequence, n, it is necessary to examine all the bits in the Randomiser repeatedly to find the largest value for n that is acceptable. For a Complete Composite Sequence the number will be the square root of length, however, for an incomplete Composite Sequence this number will be greater than the square root of the length of the sequence.

5.8.4. Determination of Basic Sequence

A Composite sequence is constructed from sub-sequences and a Complete Composite Sequence contains every possible rotated sequence in some order. A Random sequence is non-deterministic.

Given a randomiser of length 1 * 10^6 bits, which may be composite and complete it is examined to determine if it is Composite. Starting with the smallest reasonable Basic Sequence of 2 bits, determine if the 2 bit pattern is repeated in the randomiser and after a number of iterations of the test, as yet undetermined, make a decision to stop and try another sequence. Testing blocks of bits for similar numbers of zeros and ones may become time consuming even with an optimal testing strategy.

One possible method could be: for each n, select at random a number, k, of n bit samples from the sequence. Analysis of these k sequences of n bits should indicate similar values for P(0) where P(0) is the probability of obtaining a zero for any bit in the sequence.

For example, using a portion of a randomiser treated as a circular buffer, ℵ', and k = 2^n, and random vector [6, 35, 23, 28] tests can select four two bit samples.
ℵ' = 111-11101000-00011101-00111010-01110100-11010001-101

Counting from the left, the sequences selected 10,11,01,01 produce a value for P(0) of 0.375 and P(1) of 0.625. It is noted that '00' is not represented.

This test may be very difficult to implement due to the necessity of determining when to stop testing and what should be the acceptable limits on the calculated values. Additionally, whether the results of the test are conclusive or should further tests be implemented to determine the rotation sequence.

Alternatively, starting from the beginning of the sequence, select overlapping sequences of n bits. For every integer value between 1 and k discard a bit and repeat the selection. Determine the number of 'ones' in each sub-sequence and look for a match at intervals of n. If there is a match, generate every non-overlapping n bit sequence from the start and count the number of bits until the value changes. The value will be repeated for every rotated sequence of the Basic Sequence.

For example, using a sequence ℵ' and the number of blocks, k = 2 * n
note: matching numbers that are n values apart are a possible basic sequence size.
ℵ' = 111-11101000-00011101-00111010-01110100-11010001-101
n = 8
11111101, 11111010, 11110100, 11101000, 11010000, 10100000, 01000000, 10000001,
00000011, 00000111, 00001110, 00011101, 00111010, 01110100, 11101001, 11010011
7,6,5,4,3,2,1,2,
2,3,3,4,4,4,5,5 4 matches at interval 8 then try each 8 bit sequence.
11111101 00000011 *10100111 01001110 10011010 001101* = 7,2,

11111010 00000111 *01001110 10011101 00110100 01101*	=	6,3,
11110100 00001110 *10011101 00111010 01101000 1101*	=	5,3,
11101000 00011101 00111010 01110100 11010001 *101*	=	4,4,4,4,4
11010000 00111010 *01110100 11101001 10100011 01*	=	3,4,
10100000 01110100 *11101001 11010011 01000110 1*	=	2,4,
01000000 11101001 *11010011 10100110 10001101*	=	1,5,
10000001 11010011 *10100111 01001101 0001101*	=	2,5,

The Basic Sequence length is apparently 8 as the value 4 is repeated for each block in the sequence. It is also shown that the partial block size at the beginning is three bits. Appendix 9-20 contains the worked example.

5.8.5. Summary of Test

- This test identifies the correct Basic Sequence size which produces a sequence of matching values for each n bit block. Additionally, it determines the partial block size at the beginning of the sequence.
- The Basic Sequence is relatively easily determined for small values of n. The alternatives to make this more difficult are large sequence sizes or increased complexity. Large sequences will only increase the work load in a linear fashion and an optimal strategy may quickly determine the Basic Sequence size and rotation ordering.
- The work load for a Basic Sequence is approximately equal to the number of sequences generated, A, plus the number of additions, B, plus the number of comparisons, C, plus generating further n bit sequences, D, + further comparisons, E.

$$\text{Work load} = A + B + C + D + E$$

Equation 5-7

- For n = 8 the values are $A = 2 * n$, $B = 2 * n$, $C = n$, $D = c1(2 * n)$ and $E = c2(2 * n)$ where C1 and C2 are some arbitrary value. Work load = $A + B + C + D + E = 5n + c1(2n) + c2(2n)$ where a linear increase in n produces a linear increase in the Work Load.

5.8.6. Increased Complexity

A Composite Sequence can be constructed for increased complexity with multiple random Basic Sequences of varying length rotated by a random selection of values. This will break up the relationships between consecutive blocks and the solution may become non-trivial.

For example, a Composite Sequence constructed from Basic Sequences of differing lengths such as A, B, C may be ordered as specified by $\mathfrak{R}$ =[A3,C2,C1,B1,A5,B2,C5]. These sequences may be sub-sequences of a larger random sequence broken up into arbitrary sized pieces and assigned an identifier.

Given Basic Sequences A = [0101110], B = [01100110001] and C = [101] a Complex Composite Sequence, $\mathfrak{I}$, can be produced.

$\mathfrak{I}$ = 1110010-110-011-11001100010-1001011-10011000101-110

A Complete Composite Sequence for A would have 49 bits, for B, 121 bits and for C 9 bits. There would be 21 possible rotated sequences that could be concatenated 21! Times. Use of larger random sequences of the order of $1 * 10^{11}$ bits may make the task of determining the basic sequence very difficult.

Any random or deterministic sequence can be treated in like manner and it may be a non-trivial task to determine whether a portion of the sequence is deterministic (composite) or random. Any portion produced by sub-sequences smaller than a Basic Sequence may be indistinguishable from a random sequence under any reasonable test.

5.8.7. Testing a Complex Sequence

Using the eight bit DeBruijn sequence '00011101', as sequence A, '010' as sequence B and $\Re$ =[A0, A1, B1, A2, A3, A4, A5, A6].
$\mathfrak{I}$ = 00011101-00111010-100-01110100-11101000-11010001-10100011-01000111

By inspection, a repeated sub-sequence is 111010 and this may be a part of the Basic sequence. The format displayed below shows the number of 1's in the first n bit sequence and each consecutive sequence discards the leading bit. However, testing for 8 bit patterns which are expected:

4 = 00011101 00111010 10001110 10011101 0001 ... 0111= 4,4,4,5

4 = 00111010 0111010100011101001110100011010001101000110100011 1

4 = 01110100 1110101000111010011101000110100011010001101000111

5 = 11101001 1101010001110100111010001101000110100011010001 11

5 = 11010011 1010100011101001110100011010001101000110100011 1

5 = 10100111 0101000111010011101000110100011010001101000111

4 = 01001110 10100011 10100111 0100 ... 0111 = 4,4,5

5 = 10011101 0100011101001110100011010001101000110100011 1

4 = 00111010 10001110 10011101 ... 0111 = 4,4,5

5 = 01110101 000111010011101000110100011010001101000111

5 = 11101010 00111010011101000110100011010001101000111

4 = 11010100 0111010011101000110100011010001101000111

3 = 10101000 111010011101000110100011010001101000111

3 = 01010001 11010011101000110100011010001101000111

4 = 10100011 10100111 0100011 ... 0111 = 4,5

4 = 01000111 010011101000110100011010001101000111

The interpolation of a simple three bit sequence into an ordered rotation sequence hides the basic sequence structure. The test may detect the Basic Sequence depending on the application of the test and the observations drawn by the user. However, while a User may visually determine the Basic Sequence in this example, a test implemented on a computer may need to be complex.

Detailed explanation of the method is provided in the Appendix 9.21.

5.8.8. Specification of a Complete Composite Sequence

A Complete Composite Sequence can be specified in n^2 bits, however, the information needed to generate a Complete Composite Sequence is the Basic Sequence and the rotation and concatenation sequence, $\Re$. This will be n bits for the Basic Sequence plus n rotations specified in binary format. Ordering information, O,

$$O = n + (n \log_2 n)$$

Equation 5-8

Specification of a Composite Sequence in this manner is more efficient for large values of n even allowing for the work in generating the Composite Sequence. The larger the Randomiser simulated, the more efficient the specification, S, of the Composite Sequence becomes:

$$S = \frac{O}{n^2} = \frac{n + n\log_2 n}{n^2}$$

Equation 5-9

To determine the limit on S as n increases:

Using L'hopital's rule (Grossman p448) for the indeterminate form ∞/∞, divide through by n

$$\lim n \to \infty S = \lim n \to \infty \frac{(1 + \frac{\log_2 n}{n})}{n} \approx \frac{1}{n}$$

Equation 5-10

limit n → ∞ 1/n = 0

As n increases toward infinity the limit on S tends toward zero.

For example, if n = 8 then each rotation can be specified in a minimum of $\log_2 n$ or 3 bits and there are 8 possible rotation numbers in the set $\Re$ ={0,1,3,2,6,5,4,7} and using Equation 5.8 this information can be stored in a minimum of S = 8 + (8 * 3) or 32 bits which is 0.5 of the length, n^2 or 64, of the Complete Composite Sequence

A Basic Sequence of 1.37438 * 10^{11} or 2^{37} bits (17.1798698 Gigabytes) can produce a Complete Composite Sequence of 1.88894 * 10^{22} bits by specifying the rotation sequence. This Complete Composite Sequence can be specified in S = 1.37438 * 10^{11} + (1.37438 * 10^{11} * 37) bits. This evaluates to 5.22267 * 10^{12} bits (652.83375 Gigabytes) which is substantially less than the Complete Composite Sequence of 1.88894 * 10^{22} bits (2.361175 * 10^{12} Gigabytes). There are 2^{37} factorial ways of ordering the rotated sequences to produce a similar sequence.

A Complex Complete Composite Sequence can be specified by the sum of the ordering information used to generate the various rotated sequences from the Basic Sequences.

Where the Complete Composite Sequence is not used in its entirety, then to save space and time it need not be generated. For example, when only 5 * 10^9 bits are selected or discarded then only the relevant portion of the Complete Composite Sequence of 1.88894 * 10^{22} bits need be generated.

Additionally, a randomiser of N bits may be partitioned into as many smaller subsequences as required.

For example, with N = 1 * 10^6 bits then subsequence A = 1000 bits, B = 10000 bits, C = 100000 bits and D = 889000 bits, or further A = 12345 bits, B = 543210 bits, C = 123321 bits and D = 321124 bits.

The ordering of the subsequences may be changed to produce differing randomisers. For example, ABCD or ACBD. It is clear that reconstructing the Basic Sequence from an unknown partitioning of a random sequence may be difficult when little is known of the subsequences.

5.8.9. Summary

□ Any random or deterministic sequence can be rotated and concatenated and it may be a non-trivial task to determine whether a portion of a sequence is deterministic (composite) or random.

□ Use of sub-sequences smaller than a Basic Sequence may make it infeasible to determine the Basic Sequence and the Rotation order.

□ A composite randomiser can be stored efficiently in a reduced form with suitable rotation and ordering information. This storage (or perhaps 'compression') of the randomiser is more space efficient than storing the randomiser is its original form.

□ Complex Randomisers may be infeasible to analyse.

5.9. Transactions with the Smart Card Function Generator

5.9.1. Introduction

Having defined the characteristics of the Smart Card Function Generator, some of the uses need to be defined.

The Function Generator (FG) is normally considered to be unique, except when a matched set of FGs is created for symmetrical processes.

The 'simplex' implementation refers to a single unique FG used in asymmetric functions. An asymmetric function uses the unique FG physically associated with an artefact in such a way as to be readily identifiable and able to provide an authentication of the artefact.

'Duplex' or 'Multiplex' implementation refers to a matched pair or set of FGs to perform symmetric functions. Symmetric functions require that an identical copy of the data necessary for the function is available at both ends.

5.9.2. Simplex Implementation

An originator can interrogate the FG and store the input and output sequences as a means of identifying the FG uniquely. A unique FG can be bonded to an artefact or assigned to a person. Possession of specific information about the FG can be used to prove access to and knowledge of the FG and its accompaniment.

An auditor can randomly interrogate the FG and store the input and output sequences. These input and output sequences may not be inscribed on the artefact as proof that the auditor has seen it. This implies that many different auditors can take unique proofs of the seal's integrity without conferring.

An attempt at fraud may include suborning an auditor. However, introducing an arbitrary selection of auditors from a pool may complicate the task. As it is not known which auditors may have validated the FG the opportunities for fraud are reduced.

If all the input sequences used are known then a spurious FG can be implemented which can duplicate the correct response to each input sequence as a replacement for a genuine FG. However, if the auditor keeps a secret set of random input sequences and their corresponding outputs from the FG the substitution may be readily detected. This is a form of collective security in that an unknown number of output sequences may have been taken and each sequence must be repeated for each person for the artefact to be authenticated. Consequently, an artefact may be validated by an unknown number of anonymous auditors and it may not be practical to forge an FG.

Unilateral anonymous identification can be achieved by providing an FG that can be interrogated to produce an output sequence that is acceptable to a third party. For example, to prove the attendance of a person at a specific location, the person must interrogate an FG and store the input and output sequences. This can later be used as proof that the person and the FG have communicated.

Bilateral anonymous identification can be achieved by an exchange of details from their respective FGs. For example, a person can obtain the output sequence of their own FG using a specific input sequence. This can be used to interrogate the other FG to prove a meeting has taken place without the necessity of divulging personal details.

5.9.3. Multiplex Implementation

A unique matched set of FGs can be produced and an originator can interrogate the FGs and store the input and output sequences as a means of identifying the FGs uniquely.

In a symmetric system, such as access control, one card may be retained on the inside of the secure area and a matching card must be produced from the outside to validate an access request.

Additionally, the matched pair of FGs can be used in an encryption system to provide key generation and management facilities.

5.9.4. Combination Implementation

For example, a single FG may be bonded to a shipping container in such a manner as to be destroyed if the container is opened. The details of the FG output can be stored by the vendor and transmitted to the customer by separate secure communications. If Seal is validated before the package is opened then the customer can be sure that the package is the same as the Vendor has sent.

A Vendor may assign a matched FG to a shipping company which provides a courier service. This FG is compared against the retained matching FG to identify the courier as coming from the shipping company. The shipping company can issue an employee with a matched FG which can be used to identify the courier picking up a package.

The courier's details can be stored by the Vendor as proof of pickup and the courier can retain details of the Vendor's package.

5.9.5. Summary

- Identification of individuals or artefacts can be rapidly achieved with implementations of the FG.
- Identification does not necessarily require disclosure of personal details at the time of the transaction

5.10. Conclusions

5.10.1. Theoretical Method

- The theoretical method is impractical to implement fully.
- The size of the randomiser and work required for a solution are the principal constraints.

5.10.2. Proposed Method

- The proposed method can be implemented fully.
- The randomiser size constraint can be met with the deterministic method of generating sequences.
- The proposed method describes a stream cipher application that meets the recommendations of the theoretical method but is easier to implement and faster.

5.10.3. Proof of Security

- As the proposed method encapsulates the theoretical method, the proof of security for the theoretical method is not necessarily applicable.

5.10.4. Composite Sequences

- A binary sequence of any length can be constructed as a Composite Sequence by rotating and concatenating any Basic Sequence.
- The Basic Sequence can be of any length and the construction of the Composite Sequence can be specified in less bits than a Complete Composite Sequence.
- It may be difficult to determine the Basic Sequence and the rotation ordering by analysis to reconstruct the randomiser.
- It may be impractical to distinguish a portion of the Composite Sequence from a portion of a Random Sequence.

- Complexity of the Composite Sequence can be enhanced by using multiple basic sequences of differing lengths.

5.10.5. Fraud Costing

- A primary measure of the security of the proposed method is the costs and likelihood of fraud.
- The Function Generator design is practical and easy to implement.
- The Function Generator design is provably secure under single use conditions and conditionally secure under limited use conditions.
- The construction of large randomisers is practical but analysis of the characteristics of these randomisers may be impractical.
- Asymmetric and symmetric implementation methods may be used to provide security under a wide range of conditions.

5.11. Smart Card Implementation of One Way Function

5.12. Introduction.

Simulation of the Smart Card design described in the previous chapters will be addressed. The development environment is 'Turbo Pascal'. This imposes a number of constraints on the simulation such as array length and specific exception handlers but they do not affect the proposed design.

The design documentation contains a structure chart, top-level algorithm and complete source code in a separate appendix to the thesis. The implementation of the design and the test data give a basis for decisions relating to further development of the design on a Smart Card.

Development environments exist to implement the algorithm on a Smart Card but this option is outside the scope of the research. Gemplus manufacture a reader and software environment which can be used to develop a working Smart Card. (GCR400)

5.13. Design of Smart Card Application.

5.13.1. Introduction

The function generator described in chapter 5 was developed as a model for the generation of random-like sequences from a known input sequences. The use of the method to generate identification and authentication sequences is fundamental to the development of the thesis and its resolution in terms addressing the questions posed.

5.13.2. Functions

5.13.2.1. Serial Numbers and Card Identification

Each Smart Card must have a unique serial number that cannot be removed without destroying the card. This serial number identifies the card and permits it to be assigned to an individual or group. This first or external serial number is created when the card is manufactured and can be embossed or encoded on the surface of the card, for example, bar coding to permit scanning. Additionally, the serial number can be stored in ROM on the card to provide an electronic method of displaying it. This is to reduce the possibility of a chip being excised from a Smart Card and grafted onto another card with different identification characteristics.

A second or internal serial number can be generated by implementing a standard algorithm which generates an output sequence from the randomiser. Either the first sequence of bits from the randomiser can be chosen or a specific selection algorithm can be employed using a standard input sequence. This second number positively identifies the randomiser on the card and enhances the card security by ensuring that the randomiser as well as the Card is clearly identified. The hard-wiring of the serial number into the fabric of the card appears to be the best option as the manufacturer can ensure that card serial numbers are unique.

5.13.2.2. Password And Access Control

Identification of the authorised user of the card may be important depending on what function is required. For example, an electronic image stored within the card may be compared against the image of the person presenting the card.

A password to permit selection of functions is needed and this requires that either the password is unchangeable and issued with the card or the card must be field programmable either by the bank or by the user.

Limitation of the number of access operations is needed for three reasons, restricting losses, identifying illegal transactions and preserving card security.

If a misused card is reserved for low value, off-line transactions to preserve its utility, there will be little benefit from these transactions owing to the time taken to perform them. A misused card may either expire or be refused by the system operators. If the serial number is coded to identify the year and month of the issue, the access control can be designed to ignore cards without the necessary characteristics.

High value, on-line transactions need to be authorised and a check of internal and external serial numbers can identify stolen or misused cards. Biometric or imaging details stored on the card can provide a check on the identity of the user. Automatically storing the user's image at the time of the high value transaction may limit the amount of fraud.

Accessing a card by trying passwords at random can be prevented by counting the number of failed attempts and disabling the card if these attempts exceed the pre-programmed limit. A temporary lockout can be implemented where the card refuses to function for a period of time or the card can self-destruct when misused and destroy the randomiser and the selection algorithms stored on the card.

The card access-control module limits the amount of time the card can remain active without an activity being performed. This time-out function limits the amount of use to which a card can be put without the password being re-entered.

In the implementation a 3 digit serial number is used. Larger passwords may be implemented at the wish of the manufacturer or user.

5.13.2.3. Hash Functions.

A hash value of a transaction provides the User with a check on the transaction. Validation of the hash value by the User before committing a signature and confirming acceptance of the transaction ensures that the transaction is not altered.

Owing to the limited space on a real Smart Card, the hash value may be omitted from the implementation. However, an implementation that mimics MD5 operations has been used in the simulation.

5.13.2.4. Validation Sequences

The use of the card to generate a validation sequence is analogous to a digital signature where each unique input sequence generates a unique output sequence.

The serial number of the card can be appended to a document prior to hashing and the hash value could be used to generate a validation sequence. This would provide both parties with a unique signature that is related to the transaction, the consent of the User and the specific card assigned to the User.

5.13.2.5. Encryption and Decryption.

A hash sequence of a document can be used to generate an encryption key to a block algorithm such as Data Encryption Standard (DES). The validation sequence can be prolonged to provide a key sequence for a stream cipher application.

Depending on processing speed it may be possible to use the Smart Card to generate a stream cipher encryption to provide a User with on-line secure voice, mail or electronic document communication for commercial confidentiality reasons.

The key sequence can be transmitted with the message header or sent separately as the Smart Card used is unique.

5.13.2.6. Randomiser.

The randomiser is used to deterministically generate sequences in response to particular inputs. It can only be copied at time of manufacture. The selection algorithms can be altered by the manufacturer to provide *certain characteristics for sets of cards with the same randomiser or selection algorithms*. Alternatively, the Issuer or User may program the Selection algorithms for a particular card.

5.13.3. Security

The design meets the requirements of a 'One-Time-Pad' which is provably secure only if it is used once and discarded. The security is reduced by the repeated use of the method even though bits are used in different sequences. Consequently the user must determine a limit to the amount of uses permitted.

5.13.4. Memory Requirements

Assembly Language implementation of common algorithms exist and Rankl & Effing suggest that a Hash algorithm requires 4 kilobytes of code (Rankl & Effing,1997, p84) and the Data Encryption Algorithm (DEA) requires 1 kilobyte of code. (Rankl & Effing, 1997, p71)

Estimation of the total memory requirements is based on the following considerations. A stream encryption method is more space efficient than a block algorithm, bit selection algorithms may require less space than a hash algorithm and the randomiser may be as small as 10000 bits (1250 bytes).

The executable simulation program, SMARTCD.EXE, incorporating all the functions including access control, bit selection algorithms, stream encryption and the MD5 hash algorithm takes approximately 21 kilobytes of memory. This compiler produced executable file could be implemented more efficiently in assembly language.

A Philips P83C855 Smart Card has 20 kilobytes of ROM, 2 kilobytes of EEPROM and 512 bytes of RAM. (Rankl & Effing,1997, p430)

Consequently, the memory estimation is 4 kilobytes for a hash function, 2 kilobytes for a stream cipher function, 2 kilobytes for the bit selection algorithm and 4 kilobytes for access control, selection and recording. This leaves 8 kilobytes for additional applications and implementing the randomiser. For example, a randomiser of 32768 bits can be implemented in 4 kilobytes.

5.13.5. Summary

- This limitation of usage has substantial economic consequences both in the costs of manufacturing and distribution of new cards and the costs of fraud in a compromised system. The implementation costs reduce with many cards in use but the opportunity for fraud increases with the uses. For the purposes of the simulation a limit of 500 uses is chosen.
- Implementation is practical with the estimated memory usage. However, further investigation into the viability of the Smart Card design is desired.

5.14. Software Simulation of the Design

5.14.1. Introduction

The simulation environment is a Pascal development environment which provides for the development of a suite of programs to implement the functions listed.

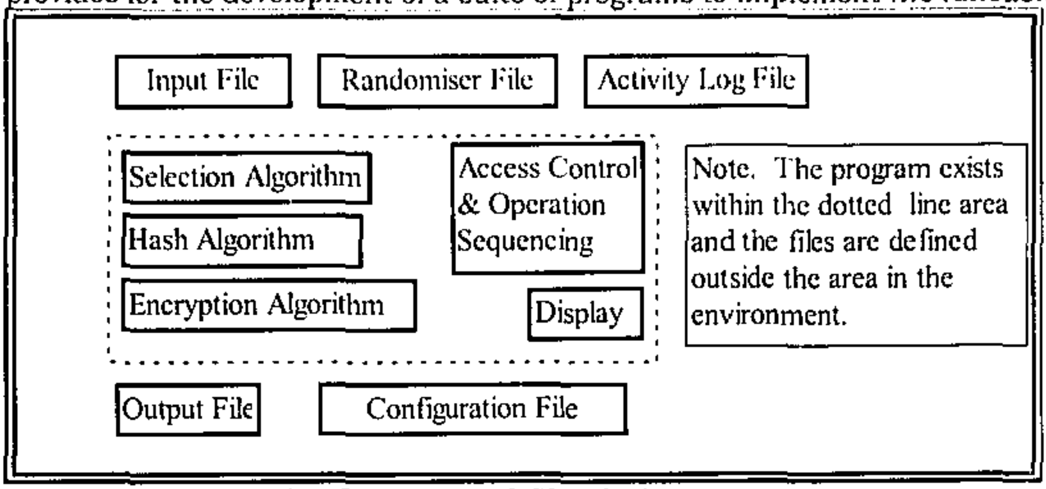

Input File
Randomiser File
Activity Log File
Selection Algorithm
Hash Algorithm
Encryption Algorithm
Access Control & Operation Sequencing
Display
Note. The program exists within the dotted line area and the files are defined outside the area in the environment.
Output File
Configuration File

Figure 5-1 Schematic of Smart Card Simulation

In figure 5-1, the inputs and outputs are either data files or keyboard inputs to permit testing and evaluation of results. A limited amount of display is available on the screen to administer the program and explain what is happening. The primary aim of the simulation is to develop data for evaluation rather than examine user interface aspects or comment on the physical characteristics of the card.

5.14.2. Functions

5.14.2.1. Serial number

The serial numbers can be generated algorithmically from the randomiser using a specific input sequence or it can be stored in the configuration file. Both options permit the card to be positively identified.

5.14.2.2. Selection Algorithm

This has been discussed in chapter 6 and is being implemented with characteristics to suit a proposed Smart Card regime.

5.14.2.3. Hash Algorithm

The 'Message Digest 5' or MD5 algorithm has been available since 1991, having been proposed and implemented by R Rivest. As this hashing algorithm has weathered the probing attacks on its characteristics it is considered to be a reasonable algorithm to use.

5.14.2.4. Encryption Algorithm

The proposed encryption algorithm is a stream cipher for speed and simplicity of processing. An implementation of DES or other block cipher, however, could be used. Symmetric algorithms require that the encryption key and the decryption key are available to the sender and recipient of the message, which means that the key is as important as the message.
This can be provided by having two Smart Cards created identical at manufacture. One can be issued to a User and the other retained by the Authoriser.

Key generation for a block encryption method can be the hash value of a text file that can be communicated by a separate method to the recipient. Stored key tables can be used to select a new key sequence and then it can be deleted. This way, a user of the system can produce a reasonable crypto-system using a simple computing package which is dependent on the availability and security of a Smart Card at both ends of the system.

Key security can be enhanced by destroying the Smart Card by exceeding the amount of permitted log-ins. Additionally, a User may destroy a card rather than permit it to be misused.

5.14.2.5. Display

The display is a simple visual indication of the current status of the card and option permitted. The effective life-cycle, measured in number of accesses, is indicated by a bar graph display and a non-functional card can be indicated by a complete graph.

5.14.3. Implementation

The executable file for the simulation is approximately 21 kilobytes not including the randomisers and data files. Implementation in Assembly language and application specific hardware may substantially reduce this requirement. Files and file access routines could be implemented by ROM access while the hash algorithm and control functions could be implemented in hardware. The randomiser and selection data can be implemented in ROM.

5.14.4. Summary

- □ The simulation is able to process all the transactions required.
- □ The display is adequate for the user to select or reject as required.
- □ It is expected that an implementation in assembly language may be much more economical in terms of memory storage. Additionally, a purpose built hardware implementation may reduce the memory requirements substantially.
- □ Implementation of an Application Specific Integrated Circuit (ASIC), while expensive, could be cost effective if many thousands of cards are manufactured.

5.15. Trials

5.15.1. Introduction

It is proposed that the software is duplicated in a simulated transaction environment to permit the testing of the design. With a PC to simulate both Smart Cards the functions and the responses can be tested.

5.15.2. Lease Protocol

Two copies of the software can be created with changes to the identifying details. The randomiser and selection sets can be changed to ensure two complete Smart Card Simulations.

Each User can then use the Smart Card to hash a sequence to obtain a value. This value can be appended to the contract and then hashed again. The hash value depends on the original hash value of the contract and the signature sequences appended to it. The card serial numbers can be appended to identify the card in the transaction as well as a signature sequence.

5.15.2.1. Testing

- □ A transaction file listing various activities and the costs involved was created and the information used as input to the Smart Card

- Comprehensive testing is not within the scope of this research which is concentrating on proof of concept rather than absolute security testing.

5.15.3. Summary

- Creation of two sets of Simulation Data is not difficult. Different responses are recorded for identical inputs.

5.16. Possible Transactions

5.16.1. Introduction

The two main classes of transactions are off-line and on-line. Off-line refers to transactions that do not require validation by the other party before processing. Ideally this will refer to low value, infrequent transactions. On-line refers to high value transactions such as bank transfers which require validation for legal and business reasons or secure transactions that must be validated before access can be permitted.

5.16.2. Off-line transactions

After the Smart Card is inserted into the reader:

S. Card1		Reader		Controller
[insert card]				
Input Seq.	⇐	Ask for Serial number		
[card identity]				
Serial No.	⇒	Input Seq.		
		[interrogate Card]		
Input Seq.	⇐	Output Seq.		
[Validate identity]				
Output Seq.	⇒	Transaction Seq.		
		[store]		
...		...		
...		...		
		Transaction Seqs.	⇒	Vendor Transactions
				[daily transactions]

5.16.3. On-line Transactions

After the Smart Card is inserted into the reader:

S. Card		Reader		Controller
[insert card]		[card detected]		
Input Seq.	⇐	Ask for Serial number		
[card identity]				
Serial No.	⇒	Response from Card		
		[interrogate Card]		
Input Seq.	⇐	Interrogation Seq.		
[Validate identity]				
Output Seq.	⇒	Serial No. +		
		Interrogation Seq.		
		[request validation]	⇒	Input Seq.
				[interrogate card copy]
		Input Seq.	⇐	Output Seq.
		[interrogate card]		
Input Seq.	⇐	Output Seq.		
[Validate identity]				
Output Seq.	⇒	[forward]	⇒	Input Seq.
				[compare validation]
		Input Seq.	⇐	Response

```
                      [ If Card Valid then
                        hash of transaction details
                        else ignore card]
Input Seq.    ⇐       Output Seq.
[ Card Accepted ]
[ Validate transaction ]
Output Seq.   ⇒       Input Seq.
                      [Hash + Validation
                      + transaction value]
                      Output Seq.         ⇒     Input Seq.
                                                [ Store Transaction Seq.
                                                  For processing]
```

5.17. Conclusions

- The design is simple, fast and practical to implement.
- Size of the software is excessive but this may be addressed with hardware and software methods.
- Smart Card implementation must address memory limitations.
- Testing is limited to functional testing or proof of concept testing.
- Substantial testing of the design may be necessary before production could be considered

6. Evaluation of Smart Card Function Generator

6.1. Introduction

This section addresses the testing and analysis of the data generated by the proposed method with conclusions. The data will be addressed in sequence and the testing methods identified and explained.

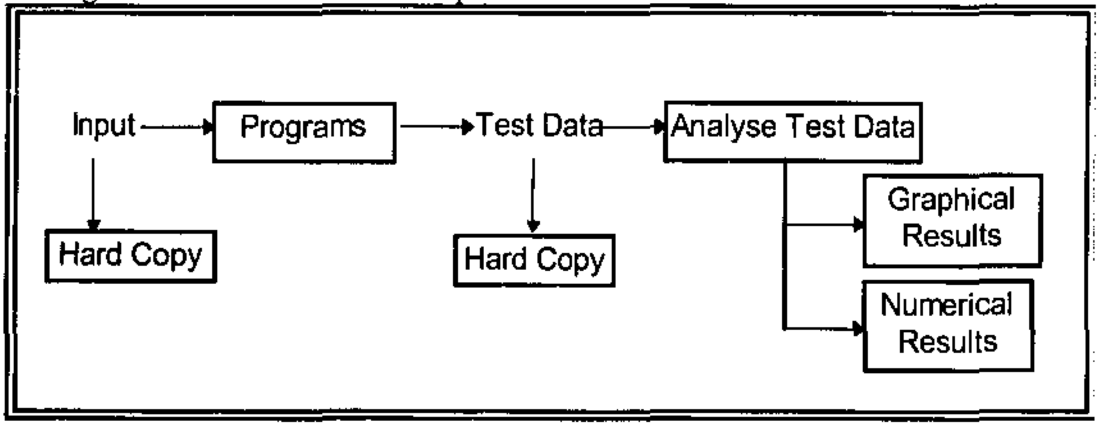


Figure 6-1 Overview of test program

Figure 6-1 describes the overall test regime. It is noted that substantial testing of all the characteristics of the design is impractical within the scope of the research. Consequently, testing of design characteristics will be discussed but will not be implemented due to time and resource constraints. The principal impediment to thorough testing is the difficulty of giving a valid proof of the security of the method. Additionally, describing randomness adequately is a non-trivial task owing to the construction of the randomisers and the methods of using them.

6.2. Overview of method

The Smart Card is able to accept an input sequence from any source and is able to calculate the output sequence according to a variety of algorithms.

A Secure Hash Function provides nominally unique output sequences from unique input sequences of varying length. The purpose of the hash function is to generate a characteristic of the input sequence which can be used to identify the input sequence and more importantly determine if it has been altered in any way since the previous hash value was generated. Consequently, it is important for the user to record the hash value produced as this constitutes a check on the validity of the input sequence which may be an electronic copy of an agreement.

Parameters which should be tested include Selection Methods for generating Start Points, Selection Sets and algorithms for generating randomisers. The Input Sequences, Output Sequences and randomisers need to have their respective characteristics determined and any encryption methods need to be evaluated.

A substantial concern is any correlation between the input and output sequences. Owing to the large amount of input sequences, in this case 2^{128} , it is infeasible to test any significant portion.

6.2.1. Selection Methods

In general selection methods must be deterministic (able to be repeated exactly) but have the characteristic of being unpredictable to any reasonable degree to ensure security of the method.

6.2.1.1. Start Points

The calculation of the starting point for the selection algorithm is by taking a selection of bits from the input sequence and producing an integer value from them. This process is not limited by the size of the input sequence providing that the input sequence is of sufficient length, which is as yet undetermined.

For example, should the input be a typical sequence from a secure hash function, (MD5 produces a 128 bit sequence) then the selection of a starting point is simply a matter of choosing and arranging some of the bits in the input sequence into a binary representation of an integer.

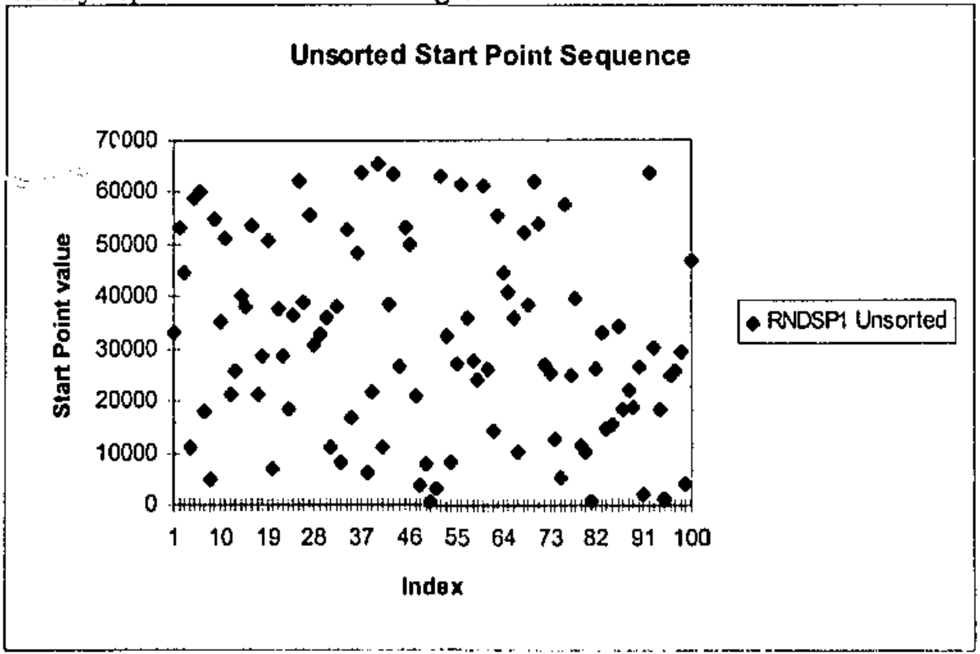


Figure 6-2 Scatter Plot of Start Points.

This selection method can be described as a set, SPI, of integers defining the order, index and number of the bits to be selected from the input sequence. The numbers in SPI are integers of any value and may be repeated producing 'an unordered multiset with repetition' (Lew p46). A reasonable practice may be selecting bits as a combination that is 'an unordered multiset without repetition'(Lewp46).

For example, with I = {01100101010100101001010} (reading from the left) and SPI = {8,3,6,1}, the start point would be '1110' which translates into 14 in the decimal base. The only limit on the length and values of S is the amount of time taken in calculating the start point as any result can be adjusted by the modulo N division of the number. (N is the length of the randomiser in use)

An initial test produced 100 16 bit start points from a single selection algorithm using 100 random input sequences. These results were stored in a file to be processed by a spreadsheet package.

In figure 6.2, it can be seen that there is very little obvious relationship between the data plotted. The scatter appears to be random with little correlation between the individual points. A simple test for relationship consists of sorting the data sequence and plotting it. A distribution that peaks may indicate a preference for a particular range of values.

After ordering the generated start points from lowest to highest and graphing them, the initial sequences showed characteristics that could be perceived as linear. (refer figure 6.3)

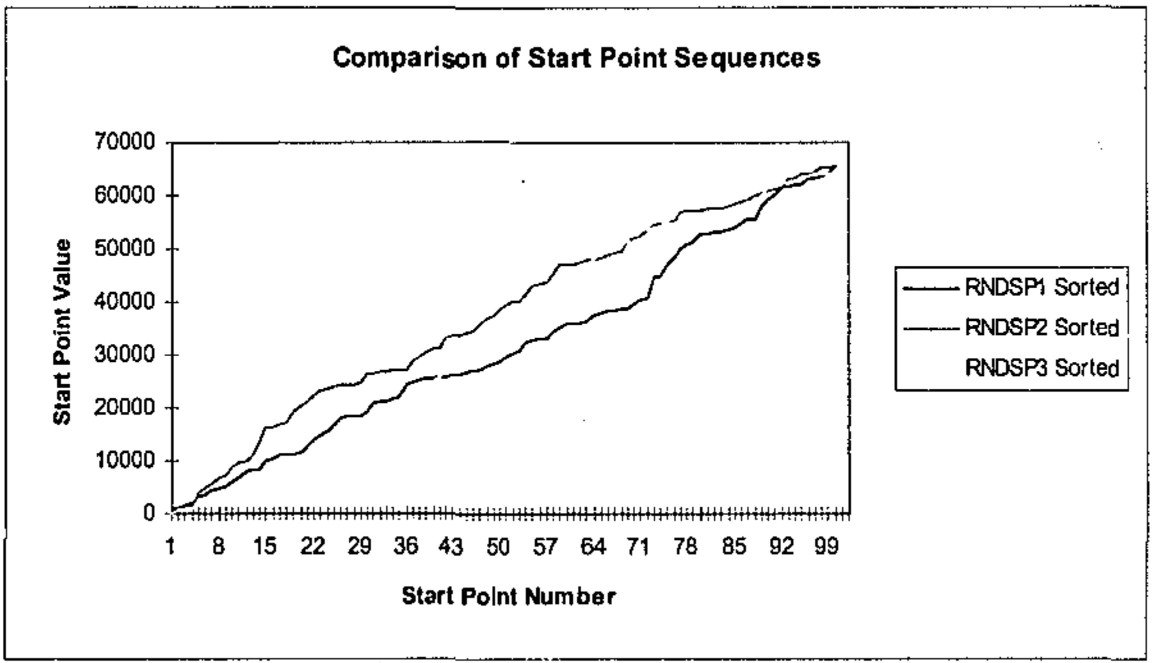


Figure 6-3 Ordered Sets of 100 Start points

The data plotted shows three sequences of start points generated from the same randomiser but with differing selection sequences. It can be seen that the data appears to have linearity and the next step was to calculate and plot the regression lines of the data.

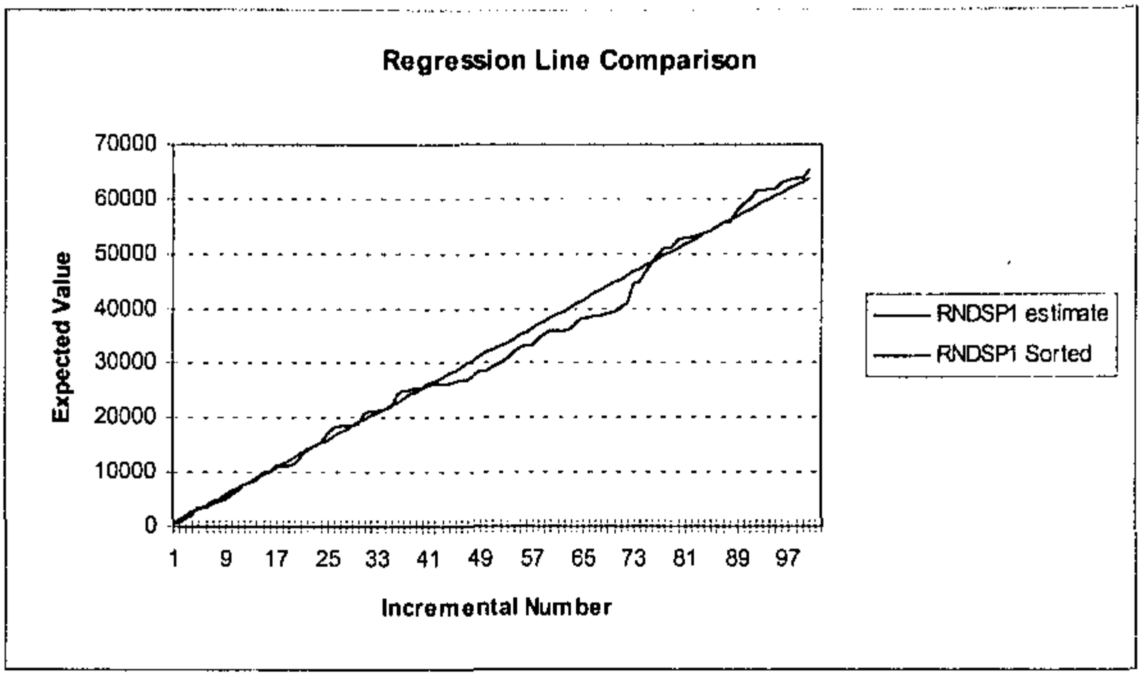


Figure 6-4 Comparison of regression estimate with sorted data

The selection sequences were:
RNDSP1.DAT = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}
RNDSP2.DAT = {16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1}
RNDSP3.DAT = {59,2,5,103,123,107,64,68,69,90,70,35,8,33,19,78}

where the bits were selected from a sample of 100 random bit sequences of 128 bits in length.

The linearity was checked by performing a linear regression on the data and plotting the derived regression line with the ordered data sequence.

The data plotted in figure 8.4 shows that the regression line has a close similarity to the ordered data and constitutes a good approximation of the data RNDSP1.DAT.

It is considered that any data with a uniform distribution across the range will display a trace with linearity after it has been sorted.

6.2.2. Selection Sets

Selection Sets are groups of integers that specify the number of bits to select or discard during the construction of a random sequence from the randomiser. The size of the selection set is arbitrary and may range from one to many thousands of integers. It is considered that a selection set could be created by expanding an exponential or factorial number and selecting digits from the sequence. For example, there are 158 digits in 100! and this factorial number may be selected by a combination of bits in the input sequence.

If the input sequence is uniformly distributed across the sequence space, that is, each possible sequence is equally likely then the input sequence is assumed to be random.

A typical input sequence may be 128 bits such as the output of a hash generator. For example, using a randomiser of 1.125899907 * 10^{15} bits (2^{50} bits) any 50 bits may be used to specify a Start Point. From the remaining 78 bits a specific selection of bits can be used to specify the selection sequence such as 13 six bit numbers ranging between 0 and 63.

Alternatively, a specific selection of 20 bits will give a number between 0 and 1048575 and this number converted into an exponential or factorial could provide a selection set of reasonable length. Additionally, the digits in the sequence can be grouped to give multiple digit numbers. For example, from the sub-sequence 123456, numbers 1, 23 and 456 can be selected.

There are many possible ways of specifying the next bit to select or discard and as long as the bits in the randomiser are random then it is expected that the samples from the randomiser are also random. Consequently, testing the selection set is not considered necessary.

6.2.3. Randomisers

Randomisers for use in this system should be as large as possible to permit the user the greatest freedom in selecting bits. The randomiser should be random or if this is not possible then it should be indistinguishable from random when it is used.

Generation of randomisers includes deterministic, non-deterministic and hybrid methods.

Testing randomisers demands a substantial investment in testing. The use of a randomiser in a cryptographic method requires that the randomness characteristic is as near perfect as possible. Analysis of this characteristic is not easily achieved and the amount of knowledge and work necessary to reduce the uncertainty is beyond the scope of this research.

A simple approach using visual methods was developed with a view to sifting the test sequences for 'non-randomness'. This method is subjective but very quickly identifies generators that may be flawed. These flaws include preferences for a particular sub-sequence and any sequence that passes these initial tests can then be analysed with numeric techniques.

6.2.4. Encryption Methods

At present the only encryption method implemented is a simple stream cipher though implementation of more complex methods is practical. Hardware implementations of DES are available and it is considered that a block cipher could be used with independent sub-keys. (Schneier p295) These sub-keys can be generated by the processing of the encryption of each block through the selection algorithm to obtain a new key sequence for the next block.

6.3. Testing

A Pascal version of MD5 was acquired and modified to run under the 'Turbo Pascal' development environment. The version from GnomeWorks was written to run under 'Delphi' and the modifications to run under Turbo Pascal were relating to string manipulation routines.

The program was modified to read information from a file of data, perform a hash to the data and write the hash sequence to a file.

An input text file, 'fred.trn' contains a short summary of transactions generated by the mythical person 'Fred'. Each line of data includes the date and time of the transaction, the name of the card holder, the signature sequence of Fred, the name of the vendor and signature sequence followed by a short description of the item and the price. Date is formatted in two digits for the day, three digits for the month and two digits for the year. Time is stored in twenty four hour format and the time zone is listed alphabetically with WA time from GMT being 'J'.

For example, '01AUG98J0900 Fred Ltd 0123456789ABCDEF Bill Bloggs ABCDEF0123456789 Commodore S sedan 3300 rego 123456 blue with grey trim $50' describes the purchase of a car by Fred from Bill on the first of august 1998 at nine am for fifty dollars.

This sequence is hashed into 'AAE27E40389335F628CC0896956F7766' and this value is unique to the transaction as listed. The 32 characters are a hexadecimal transcription of the numeric 16 byte hash value.

The hash value of the transaction is processed by the Smart Card to produce an output sequence '99D16D30278224E517BB0785845E6655' which can be retained by both parties as an authorisation of the transaction. Additionally it can be provided to the bank as a means of validating the transfer of funds.

6.3.1. Testing of Output Sets

The initial concern with generating output sets from a randomiser is that the selection method does not alter the samples in any detectable way. This is to demonstrate the 'Law of Large Numbers' of statistics where the characteristics of a large group of samples from a population has the characteristics of the population.

Using an eight bit binary sequence from which any four bits will be selected as an example. To generate all the possible samples from zero to eight bits the complete range of the binary number is generated from 0 to 2^8-1 (0 to 255). These are ordered according to the decimal equivalent. Sorting this array of binary numbers by the number of ones in the sample and the decimal equivalent gives a listing of all the possible samples.

number of bits sampled	no of samples	cumulative total
0	0	1
1	8	9
2	28	37
3	56	93
4	70	163
5	56	219
6	28	247
7	8	255
8	1	256

Table 6-1 Enumeration of Samples

For a complete table see the Appendix 11.18. The table demonstrates that the average of all the samples is equal to the average of the population in accord with statistical theory.

6.3.2. Summary

- ❑ It is accepted that the average of the all possible samples from a population is the same as the average of the population. With this established, using the average of a population of samples may characterise an unknown random sequence.
- ❑ Using a set of randomisers of 10000 bits where a sample of 128 bits is to be selected, the average of the randomiser is calculated and the average of a set of 128 bit samples is calculated and compared.
- ❑ The use of a set of starting points with each selection set ensures that the sequences are comparable, that is use the same randomiser and selection sequence with only the starting point varying.
- ❑ It is noted that the average of the samples closely approximates the average of the randomiser and this supports the view that the sampling method does not alter the characteristics of the samples.
- ❑ It is noted that a randomiser drawn from a non-deterministic source may display similar characteristics.

6.3.3. Encryption Decryption Methods

It is noted that any message from a message processing system is able to provide an eavesdropper with information about the message and the system. This 'aging' parameter of the system is related to the amount of use of the system and not just the amount of bits generated.

For example, a small number of very long messages may give away less information than a large number of small messages. This is due to the structure of the messages which may have similar components such as a business salutation or an address block.

However, if reasonable limits are placed on the use of the system, the ease with which a new system can be provided is able to counter the problems of overuse or compromise.

6.3.4. Summary

- ❑ Testing may become a major task involving much effort, consequently much planning is needed to ensure that the results gained are useful and meaningful in the context of this program.

7. Recommendations and Conclusions

7.1. Introduction

This section lists a selection of possible uses for the Smart Card Design. It is considered that the design may be useful in applications other than those listed.

Conclusions are listed with reference to the original questions in Chapter 4.

Directions for future research are limited by the lack of validation of the developmental method. It is practical to implement but without substantial testing and certification if cannot be used in commercial products.

7.2. Recommendations

7.2.1. Driver's Licence

A unique Smart Card may be personalised and assigned to the Driver. Personal details recorded on the card can identify the lawful user and the electronic serial numbers can be used in automated data processing systems.

For example, a driver being booked by a Traffic Patrol may be identified by the personal details on the Smart Card. The card can be identified to the system by the unique details and the information relating to the traffic offence can be validated by storing the output sequence which would provide a signature and an authentication of the identity and transaction. Should the card permanently store details of an offence then it would be available off-line to any authorised reader.

7.2.2. Automotive Lease/Lend environment

The use of the Smart Card as a key to validate the identity of the user is practical. Once the owner of the vehicle has identified the person concerned a unique Smart Card can be issued that is acceptable to the car computer as permission to use the vehicle.

The owner ensures that the VMU can recognise the Smart Card. This can be achieved by ensuring that the VMU has a copy of the randomiser, selection, sets, and startpoint algorithms and can apply them only to validate a smart card.

During manufacture of the Smart Card, a matched set of 2 or more chips can be created and one can be hardwired into the VMU. The remaining chips can be encapsulated into Smart Cards to provide a set of Master Keys to suit the particular VMU.

This implies that the Smart Cards are uniquely and specifically related to a particular VMU and they can be considered to be a 'certificate of ownership' of the VMU and, by association, the car.

To ensure the security of the method, the VMU must be physically secure against attempts to change the 'Key' chip. In cases where the VMU is encapsulated in tamper-resistant materials it may be necessary to change the VMU for maintenance. The VMU may also include WORM memory which can be configured with specific information such as the engine number or owner details which can provide theft protection by identifying the components of the car.

The Smart Cards, as a certificate of ownership can be used as Master Keys to permit a User to access all the functions provided by the VMU. This function can be used to 'teach' the VMU to recognise another Smart Card which functions as a limited session key for a user in a lease/hire environment.

7.2.2.1. Complete Copy

If the VMU has a complete copy of the data, it can periodically ask the Smart Card to prove its identity by asking 'If you are Card XYZ then what is your answer to Question ABC' and wait for a response. The VMU will generate it's own answer after the challenge and response and compare them. If there is a match the smart card is acceptable and the use can continue. If there is no match the VMU can repeat the interrogation until at some stage the transaction is stopped.

This may detect substitution of an alternate card after the authorised card is removed.

The Owner generates two copies of a subordinate Smart Card and installs one subordinate Smart Card in the VMU which is unable to be removed without damaging it.

The Master Smart card can authorise the VMU to store identification details of the subordinate Smart Card in case it is altered or swapped. With this copy of the subordinate Smart Card installed, the VMU can then validate any access and use of the vehicle with the other subordinate Smart Card.

7.2.2.2. Incomplete Copy.

The VMU keeps a table of questions and responses from the Smart Card and selects from it at random to validate the User's Card.

This method may not be very secure, but further research may resolve this issue.

7.2.3. Electronic Commerce Environment

7.2.3.1. Large transactions on-line by default

If minimum and maximum amounts are defined for a transaction then a Smart Card can be used to authenticate transactions. Transactions below the minimum limit can be processed off-line to permit the use in all conditions. Transactions above the maximum limit can be recorded but not processed to provide evidence of attempted misuse.

All transactions above the minimum limit must be processed on-line to minimise the possibility of fraud. The security can be enhanced by the bank interrogating the Smart Card during the transaction and ensuring that the card answers multiple interrogations to limit the possibility of falsely identifying a card.

All transactions above the daily transaction limit can be validated by a Bank Employee where the image of the user is stored and compared against the details of the person presenting the card. This may become a mandatory requirement for transactions above a certain limit.

Fraud would be possible if the smart card could be copied by recording and playing back the answers to particular interrogation sequences. If the interrogation sequences are random then the chance of predicting one interrogation sequence is small. If multiple random interrogations are used then the Smart Card must generate multiple complete replies. If the reply is 128 bits long then three sequences would be 384 bits (48bytes) enhancing the security of the method.

The attempted fraud must predict beforehand what the three interrogation sequences are and generate the correct responses to return. This can be only accomplished by determining the structure and contents of the Smart Card and the startpoint, selection set and randomiser used.

Examination by recording all the inputs and outputs is impractical due to the amount of time necessary to obtain enough information to reconstruct the function generator. Sophisticated tools to examine the internal structure of the Smart Card may not be generally available.

Restricted access to the card would be achieved either by limiting the amount of uses or by limiting the amount of time before the card is declared lost or stolen. A card reported lost or a card subject to invalid access attempts can be incapacitated. A card reported lost and incapacitated then found again must be reactivated by the legitimate user.

A card may be incapacitated either completely or partially. Partial incapacitation may mean that the card can perform a limited number of low value transactions such as buying a public transport ticket but not perform transactions above a certain limit. This would permit a legitimate user to use a card until it can be re-validated at a bank office and reinstated.

If the card is stolen and the pin number is also obtained, then once the card can be reported stolen then all transactions with it can be stopped. Additionally, on-line transactions could be used to trace the location of the card and off-line transactions would limit the amount of losses incurred.

7.2.3.2. Small transactions off-line by default

With small transactions, below a specified limit , the vendor can record the details. Each transaction would be processed at the end of the day gathering these small amounts of data.

Transactions less than a specified amount can be performed off-line. A Card used to pay for public transport usage would generate identification and output sequences in response to a question. The question (input sequence) could identify the date and time of the transaction as well as the costing details. Periodically, the transport company can process the stored information and update its balances with the banks.

A public transport corporation may issue these cards as an account card similar in use to a credit card with a monthly statement being issued.

7.2.4. Access to Secure Areas

A pair of Identical Cards can be produced during manufacture. The issuing authority keeps one and issues the other to an identified person. The retained card can be kept as a reference library for the computer controlling the access systems.

Once the authorising card is removed from the library then the identification card cannot be used to access the system. This may prevent a stolen card from being used to unlawfully access premises. Should an employee leave the company without returning the card it poses no threat to the security of the system as long as the issuing authority removes the authorising card from the library.

The pair of cards must be functionally identical but may have different sizes. The identification card may be the usual size for an identification document and the authorising card may be physically smaller to fit a card frame. However, if both cards are identical in shape and have the photograph of the user imprinted then the card frame may provide a visual reference to the authorised users.

7.2.5. Identification.

Use as an identification artefact depends on being able to link the card with the user. Imprinting an image of the user on the card exterior can readily be achieved but the use of digitised images may take up too much of the Smart Card's memory. A method mentioned in 'New Scientist' suggests that a method of encoding a face into about 50 bytes of data may be suitable for use on magnetic stripe cards.(New Scientist) This could be implemented easily in a Write Once Read Many (WORM) memory location permitting a Smart Card to store an image of the user internally.

The User's face is recreated by superimposing digitised images from a library of 290 faces. The library of images is not stored on the card. The composite face is specified by the images used to construct it and this sequence is stored on the card.

This may add to the security of using the card by permitting a picture of the lawful user to be displayed for comparison or record keeping purposes. Additionally, unless a universal standard library is used it may not be possible for a hacker to recreate or modify an image without the original database.

7.2.6. Link Encryption in noisy environments.

Using a pair of cards as a link encryption method is practical in an environment where noise can corrupt a signal. Cable-less transmissions such as infra-red remote transmitters could use encrypted sequences of commands or instructions. Any sequence that does not decrypt to a valid message would be ignored by the system until it is re-sent and decrypted correctly.

If the sequence is encrypted differently a number of times then the receiver must ensure that it decrypts to a valid command before responding. This ensures that cross-talk and interference from other transmitters in the area has a minimum effect.

Superimposing two or more dissimilar signals at the same frequency may produce a detectable sequence but not every receiver may decode the signal sensibly. If the Pulse Repetition Frequency (PRF) of transmitter A differs slightly from the PRF of transmitter B then the production of erroneous signals would be reduced. Changing the fundamental frequencies of both signals would enable the erroneous signals produced by beating to be detected.

In figure 7.1, signals A and B are superimposed to produce signal C. Signal C may be a valid sequence subject to misinterpretation by the system. If Signal A and Signal B are encrypted differently a number of times in sequence then the occurrence of a meaningful superimposition would be less likely.

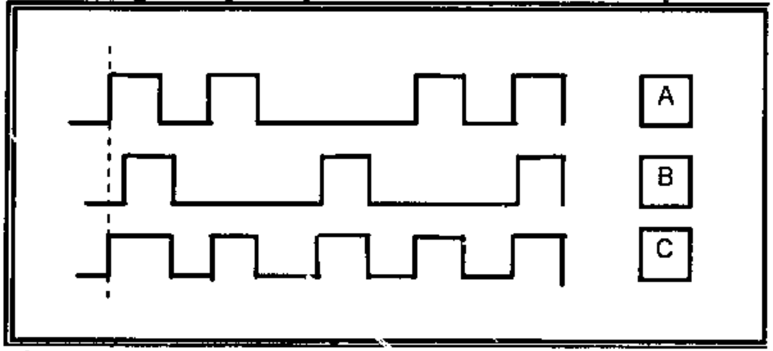


Figure 7-1 Superimposition of Signals

7.2.7. Stream Cipher Application

A development of the stream encryption method is to use larger randomisers such as families of compact discs and the rotation and concatenation algorithm. This is a most effective method of generating and encrypting a message.

7.2.8. One Time Pad

A Smart Card FG may be used to generate a key sequence for a cryptographic algorithm. As long as the FG is only used once then the method is provably secure. A single FG may be used as a private key to an asymmetric method of encrypting personal data for storage. A matched set of FG's can be used for secure communications in a symmetric method.

7.2.9. Digital Photographs

A digital image may be authenticated by a matched pair of Smart Cards. The image and its hash value can be appended to the card. The hash value input sequence can be stored with its output sequence as protection against the image being changed unilaterally.

7.2.10. Documentation verification

A function generator may be bonded into or on a hard copy of a legal document in such a way that removing or tampering with it will destroy its functions. The function generator may then provide an output sequence to the document's hash sequence and this Seal Sequence can be inscribed in the document signature or seal portion.

The originator of the document can retain the random input and output sequences of the FG to identify the artefact. This prevents the document being altered without knowledge by substituting a new document and FG. If the FG fails to function or produces an incorrect sequence then the document may not be acceptable. However, the document can be authenticated by the original issuers and a replacement issued.

Hierarchical signatures are where a supervisor signs a document prepared by subordinates who have identified their contributions to the document by a signature.

To sign the document, the seal sequence and the serial number of the respective FG are concatenated and the hash sequence is computed and may be inscribed on the document. The hash sequence produces an output sequence from the signer's FG which is appended to the signature portion.

The Hash sequence and Output Sequence then constitute a 'signature' or Seal sequence on the document attesting to its integrity. The document can then be tested for alterations.

An auditor can regenerate the hash sequence and ensure that the FG attached to the document has produced a specific output sequence. This hash sequence can be used to generate an output sequence from the auditor's Seal which can be appended to the document.

A document may be stored on a Smart Card in permanent memory. The hash value of the document can be used to generate an output sequence from the FG to provide a validation of the document.

Additionally, the VLSI or other implementation of the proposed algorithm may be incorporated into the structure of the document in such a way that tampering with the document will destroy the device.

It is considered that a 'Seal' may also be associated with a position rather than a person in the manner of corporate seals

7.2.11. Security Seals

Additionally, VLSI or other implementations of the algorithm may be incorporated into artefacts as a 'Seal' or proof mark. For example, a Smart Card or other implementation could be permanently bonded across the gap between a container and the container door. Opening the door would destroy the Seal and replacing it would be very difficult when only the legitimate user knows the details of its construction.

7.2.12. Anonymous Identification

Unilateral Anonymous Identification can be achieved by using a unique FG. Any person able to produce an input and output sequence for the FG is able to prove that they have had access to the FG. Personal details are not necessary.

For example, a security patrol supervising a set of premises must be able to prove attendance at specific times. The patrol uses one FG to prove they have attended the premises by allowing the security system on the premises to interrogate the FG and store the results. This may also be reciprocal in that the patrol can interrogate the security system FG and store the results.

For example, an artefact may use an attached unique FG for identification and its responses to interrogation can prove that the artefact has been seen by an auditor. Every auditor can randomly interrogate an FG and store an output sequence and this means that each auditor can obtain a unique 'proof' of the Seal's integrity which need not be shared or made known.

For example, a password may require a specific number of consenting 'signatures' for access to bank accounts. Each person can then take the account identification sequence and process it through their personal FG. These output sequences can be used as a password directly or they can processed provide an input sequence to the artefact's FG and if this matches the password then the access is permitted.

Bilateral Anonymous Identification can be achieved with two unique FG's. For example, two parties may use the FG output sequences to identify themselves to each other and these details can be provided to others as proof that they have communicated.

Knowledge of the input and output sequences of an FG can be used to identify an unknown person. For example, a treasure hunt where the prize is an FG which has had its details stored by the convenor of the hunt prior to it being hidden. If the correct FG is found then it can be verified for prize giving purposes.

For example, multiple copies of an identification Smart Card may have the image of the user embossed on it. If the card has an access password to initialise it then it can be used to authenticate access requests. The user initiates the access request by placing the card in a reader which may be accompanied by the user's password. The secure access control system identifies the Smart Card by its serial number and supplies a random interrogation sequence to the card and stores the response. Comparison of the output sequence with the output sequence from the copy of the card in the secure area can be used to approve access.

7.3. Conclusions

What are the characteristics of a Users, F, Resources, G, and levels, H, access environment and which authentication model, if any, could effectively represent this relationship in a real-time, distributed computing environment?

Each relationship is characterised by a triplet (F, G, H) which describes the User, Resource and Access level. A manager must identify the user and define the resources and levels of access appropriate. A single user may have multiple access triplets. The triplets (F1,G1,H1), (F1,G2,H2) identify multiple resources and access permissions. The triplets (F1, G3, H3), (F1, G3, H4) identify the different options within a specific resource. For example, for accounting purposes where one triplet describes work use and the other describes the personal use.

The authentication model proposed is based on the 'Challenge - Response' model and addresses the identification and authentication of the individual users, resources and constraints where the User or Resource is in possession of information which must be provided on demand. A User is identified by the Owner who issues an artefact which identifies and authenticates the use of the resource. Identifying the user to the resource is effected by the Smart Card which serves as an identification and authentication artefact. If the Smart Card is unique it is practical to assign a specific card to a User and subsequently identify the User and all transactions made with it.

The distributed environment may entail on-line and off-line transactions which will demand secure, reliable transactions. The use of the 'Challenge - Response' model in a transaction record enables all involved in the transaction to obtain a unique record.

1. Positive identification of the Smart Card and the Authorised User of the card is achieved by creating a unique Smart Card. The card characteristics are recorded with the details of the authorised User and are used to validate transactions both off-line and on-line. Secure transmission and storage of information will minimise the possibilities of fraud or breach of privacy. The unique characteristics of the Smart Card can be used to provide cryptographic protection of all data in the system.

2. Authentication of a User can be carried out by verifying the identity of the Smart Card. The embossed external serial number of the card may be a first-line identification and the second line could be an internal serial number.. Additionally, the Smart Card may provide specific answers to varied questions in a 'Challenge - Response' model, where the card must provide specific answers to a varied set of questions.

3. Validation of Transactions can be performed either on-line or off-line. The hash code of the transaction summary can be used to stimulate the Smart Card to produce a unique sequence which can be used to prove that a particular transaction was authorised by a specific Smart Card. Security of the method needs to be carefully examined. Initial data indicates the method is provably secure under restrictive conditions and conditionally secure under the most likely conditions.

4. Investigation into the requirements of authentication and digital signatures to determine the technical considerations that may have a bearing on the security of the proposed model.

5. A digital signature must be substantially secure against fraudulent actions and very difficult to forge. Any signature algorithm must be provably one-to-one and unique. A unique Smart Card can embody a signature generation algorithm that can ensure the signature generated is particularly unlikely to be generated by another card with the same input. Smart Card construction techniques may make each Smart Card unique. Additionally, determining the characteristics of the card by disassembly should be destructive.

6. Implementation strategies must include secure manufacture and distribution of new cards.

User strategies must include card identification methods, personalisation methods and secure access methods. The existence of a global network of electronic information systems makes the proposal effective. Cheap, bulk manufacturing of cards with variety to suit different applications.

❑ *What requirements are necessary for untamperable electronic data storage for short and long term storage to assist in resolution of disputes in the transaction cycle?*

1. Memory devices must be physically and electrically robust and securely installed. The data may be encrypted prior to storage to prevent the data being intentionally altered by illegal access. Altered data may not be decrypted in a meaningful way

which would indicate that data has been altered. Encryption makes the problem of intentionally altering the relevant data bits substantially difficult.

2. Memory access methods must be secure to permit access for reading or writing by authorised users. Audit of Data to determine its value and storage needs. Data must be auditable and able to be validated by the card and the user. Flash Memory is considered suitable for this proposal due to its long term power-off data retention (20 years). Low Power versions may use as little as 3 volts where the vehicle electrical system provides a nominal 12 volts.

3. The card must be resistant to misuse and abuse. Probing with sophisticated electronic devices must be very difficult after construction and personalisation. Data must be encrypted with an algorithm as strong as necessary to ensure that the data is secure in storage and transmission. A stream cipher would be suitable depending on the qualities of the key generation method.

4. Access to the card may be from password / PIN number retained by the user. There should only be one password for each card at a time. A manufacturer should not implement any global passwords for a card series as this would weaken the security of the system. Personal data kept on the card in secure memory and any information that uniquely identifies the user may be used. A photographic image of the user may be stored for recording and comparison by the vendor during certain transactions.

5. The amount of data to be stored depends on the construction of the prototype systems. This depends on the amount of sensors, rate of data reading and the methods of data processing. Eg. Compression, encryption etc. Business requirements may require data to be stored for at least seven years for taxation and audit purposes

6. Physical threats such as intentional or accidental damage where the memory chips need to survive a major accident and fire where the vehicle is completely destroyed. Electrical faults intentional or otherwise may affect the integrity of the data. Static electricity and high voltages from ignition systems may corrupt data or damage the memory chips. Electronic threats include hacking against the administration system. Eavesdropping on the data by unauthorised persons can be countered with secure transmission techniques which include cryptographic protection.

7. What data from the operation of a prototype is of importance in effecting the proposal and which security issues need to be addressed?

8. Memory requirements, speed of operation and reliability data is essential. Identification of the user prior to issuing the card and identification of the card in use. Authentication of each use of the resource. Secure storage of all data. Memory size and type, processing speed and data transfer speed.

9. Algorithms chosen to implement the system. Externally, embossing and proprietary data such as logos. Internally, serial numbers and access algorithms. If the issuer interrogates the card with an random sequence, the reply must match the issuer's generated reply or the card may be considered unacceptable and any transaction could be ignored. After identifying the User by the Issuer, a card can be programmed with personal details of the User. A picture may be embossed on the outside and stored inside as well. If a unique Smart Card is assigned to a particular User then the User is identified by the Card details.

10. The use of memory technologies such as the WORM memory may permit an issuer to hard-code the identification details such as a picture of the user. Ease of manufacture, update and modification where a manufacturer or issuer should be able to personalise a number of cards. A matched set of cards may be used as a party key.

11. Large scale manufacture could bring the costs of the Cards down and permit many additional uses. Reliability where a Card must repeatedly generate the same output

sequence for the same input sequence. Controls on the Card should be simple to use.

12. Positive identification of the card by the internal and external characteristics. The user is identified by the additional details stored on the card and the knowledge of privileged information such as the password of the card.

13. A stream cipher may be efficiently implemented in the limited space available and must be at least conditionally secure. The proposal is provably secure if it used only once and discarded. Multiple use means that the stream cipher is conditionally secure. The design meets this requirement. The proposal may be used to generate a key for a block cipher such as DES or may be used to provide public key details, accessible only to the authorised user. Both stream and block methods may be implemented on a Smart Card, however, the limitations are data transfer rate and memory size, coupled to processing speed.

14. The proposed method can be implemented on the Smart Card and may require less than 3 kilobytes of data depending on randomiser size. Measurements of the security are complex and may be substantially outside the scope of this research. However, the method is provably secure under single use and conditionally secure under multiple use conditions.

15. Use of proprietary protocols to identify the user before the card is issued are practical. For, example, SET and Lite-SET exist to handle funds transfer. User identification can be broken down into artefacts and information. Artefacts such as Smart Cards with unique characteristics can be recognised by the system as being genuine and acceptable. Information includes pin number , image matching and possibly even biometric methods of identifying the user. The use of hash value and identification sequences from the smart card are used to positively identify a transaction. The hash value is input into a function generator and the output sequence from the card is stored. The storage of these two pieces of information can be used to check on the validity of a copy of the transaction.

16. Secure transmission methods as all data must be transferred and stored securely. As all system information may be of use to the unauthorised person the data must be either validated or encrypted to ensure that modification or error has been detected before use. Unique Smart Cards or sets of Smart Cards matched at time of manufacture.

17. Protocols to identify users and transfer funds are generally outside the scope of this thesis. The transaction descriptions list the basis for a protocol design which would need development and formal proof before it could be used. A unique Smart Card used as a Function Generator (FG) provides a method of generating a unique output sequence from a unique input sequence.

18. The input sequence may be the hash value of a transaction. The output sequence is unique to the FG and related to the input sequence. The output sequence can be recorded as an authorisation for the transfer of funds.

19. As the Smart Card issuer retains a copy of its details, the start point, selection algorithms and the Randomiser, then reconstruction of any output sequence is possible by applying the input sequence. Transaction details, hash value and output sequence are used to recreate the output sequence for comparison with the supplied sequence. A match between the supplied output sequence and the recreated output sequence means the transaction can be approved.

20. Every party to an agreement may use a FG to generate unique sequences. Subsidiary sections of the contract can be signed with a particular sequence from its contributors and all participants may keep the final hash value. If a disagreement arises the contract must be hashed again and compared against the stored hash value. Identification of a Smart Card by its serial number and its response to specific interrogation sequences permits the Smart Card, and by association its User, to be identified to other parties.

7.4. Directions for Future Research

The research has shown that it is practical to produce a Smart Card to provide authentication, identification, secure communications and data logging. Directions for further research are constrained by one key factor, namely, the evaluation of the level of security provided by the developmental algorithm.

While the goal is 'unconditional' security, many commercially acceptable algorithms are 'computationally' secure in that the data, processing or storage requirements and complexity of the algorithm are infeasible. (Schneier pp8-9) 'Provable Security' depends almost entirely on complex theoretical mathematics as the magnitude of the analysis may exceed even the most remarkably equipped researcher. Consequently, this is outside the scope of this Master's thesis and may pose substantial questions to more experienced researchers.

Should the developmental algorithm provide provable security in the sense of being acceptable for use in Electronic Commerce then development of the suggested applications are described in the Recommendations section.

8. References

Agnew, G.B. (1987) Random sources for cryptographic systems Advances in Cryptology - Eurocrypt87 edited D. Chaum and W.L. Price Springer Verlag

Barker, W.G. (1984) Cryptanalysis of shift register generated stream cipher systems Aegean Park Press

Bassein, S. (1996) A sampler of randomness American Mathematical Monthly June July 1996

Bosch93 (1993) Automotive handbook Robert Bosch GmbH, Stuttgart 3rd Edition

edited U. Adler

Bosch94 (1994) Diesel fuel injection Robert Bosch GmbH, Stuttgart 1st Edition

edited U. Adler

Caelli, W., Dawson, E., Neilsen, L. & Gustafson, H. (1993) CRYPT-X Queensland University of Technology Brisbane

Cantrell, T. (1997) Serial Flash Busts Bit Barrier Circuit Cellar INK no. 85 August 1997 pp82-85.

Caron, O., Cordonnier, V., Durif, P. & Grimonprez, G. (1994) New Architectures for Smart Cards : the OCEAN Approach [CD-ROM] IPO (95-3)

Chandor, A. (1981) The Penguin dictionary of microprocessors UK, Penguin books.

Chaum, D. (1991) Numbers can be a better form of cash Computer Security and Industrial Cryptography (edited Preneel, B., Govaerts, R., Vanderwalle, J.) Springer-Verlag

Chang, C.C., Hwang, R.J. & Buehrer, D.J. (1993) Using Smart Cards to Authenticate Passwords [CD-ROM] IPO (95-26)

Circuit Cellar (1997) Circuit Cellar INK no. 85 August 1997

COA (1996) Review of policy relating to encryption technologies Attorney General's Department; Australian Government Publishing Service

Cooke, J.C. & Brewster, R.L. (1994) Cryptographic algorithms and protocols for personal communication systems security [CD-ROM] IPO (95-16)

Cooper, W.D. & Helfrick, A.D. (1985) Electronic instrumentation and measurement techniques (3rd. Ed.) Englewood Cliffs, NJ (USA), Prentice-Hall.

Cordonnier, V.M. (1991) Smart Cards: Present and future applications and techniques Electronics and Communication Engineering Journal October 1991 pp207-212

Denning, D.E. (1982) Cryptography and data security , Reading, Mass. (USA), Addison Wesley .

De Schutter, B. (1991) Trends in the fight against computer-related delinquency Computer Security and Industrial Cryptography (edited Preneel, B., Govaerts, R., Vanderwalle, J.) Springer-Verlag

EA (1997) Recent developments in Smart Card technology article in Electronics Australia September 1997 by Dr. J.P. Benhammou

GCR400 (1998) [on line]: URL http:// www.gemplus.fr/ products/ hardware/ index.htm (April 2, 1999)

Gibson, A. & Fraser, D. (1995) Commercial law Longmans Australia

Glass, G.V. & Stanley, J.C. (1979) Statistical Methods in Education and Psychology . Englewood Cliffs, NJ (USA), Prentice-Hall

Gnomeworks (1998) MD5 pascal implementation

http:// advicom.net/ ~gnome/ Programming/ Source%20Code/ source%20code.html

Gregory (1990) EFI and engine management - A comprehensive guide to diagnosis and repair 1st Edition 1st reprint Gregory's Scientific Publications Universal Press Sydney

Grossmann, S.I. (1987) Calculus 4th Edition Harcourt Brace Jovanovich

Hanaoka, G., Zheng, Y. & Imai, K. (1998) LITESET: A light-weight secure electronic transaction protocol Information Security and Privacy edited C. Boyd & E. Dawson Springer Verlag 3rd ACISP Conference Brisbane.

Hands, D. (1997) A design interfacing the MC68HC11 to the AMD AM29F010 flash memory chips Honours thesis for Bachelor of Engineering Edith Cowan University, Perth, Western Australia.

Harrop, P.J. (1994) The phantom tollbooth - charging for road use IEE Review January 1994

Hendry, M. (1997) Smart card security and applications Artech House, Norwood Massachussets

IEEE92 (1992) Contemporary cryptology - The science of information integrity Edited Simmons, G.J. IEEE Press, Piscataway, NJ

Johnsonbaugh, R. (1984) Discrete mathematics MacMillan

Jones, H.W.E. (1998) The use of Smart Cards in vehicle management MSc thesis Edith Cowan University, Perth, Western Australia.

Kahn, D. 1996 The codebreakers - The story of secret writing Scribner . New York

Kaye, B.H. (1993) Chaos and complexity: discovering the surprising patterns of science and technology VCH Verlagsgesellschaft mbH Weinheim Germany

Konigs, H.P. (1991) Cryptographic identification methods for Smart Cards in the process of standardization IEEE Communications Magazine June 1991 pp 42 - 48

Knuth, D.E. (1973) The Art of computer programming vol.2 seminumerical algorithms 2nd Edition Addison-Wesley

Kruse, RL. (1987) Data structures and program design (2nd Ed.), Englewood Cliffs, NJ (USA), Prentice- Hall International

Lew, A. (1985) Computer science : A mathematical introduction. Englewood Cliffs, NJ (USA), Prentice Hall.

Longley, D. & Shain, M. (1987) Data and computer security - Dictionary of standards concepts and terms Macmillan Publishers

Maurer, U.M. (1990) A provably-secure strongly randomized cipher Advances in Cryptology - Eurocrypt 90 Edited IB Damgard Springer-Verlag

Maurer, U.M. (1991) A universal statistical test for random bit generators. Advances in Cryptology - Crypto90 proceedings Springer-Verlag pp409-420

McLeish, J. (1991) Number from ancient civilisations to the computer Harper Collins

McNeil, D. & Freiberger, P. (1993) Fuzzy logic Simon & Schuster

Menezes, A.J., Van Oorschot, P., & Vanstone. S. (1997) Handbook of applied cryptography CRC Press.

Motor56 (1956) The motor electrical manual 12th edition 2nd Impression Temple Press London.

Motor63 (1963) Motor's auto repair manual 26th edition Motor Books. New York

Munro, J.E. (1992) Discrete mathematics for computing Thomas Nelson Australia

New Scientist (1998) Saving face - clever image processing is putting pictures on cash cards New Scientist 29 August 1998 Volume 159 No. 2149 article by Mark Ward p 9

OD97, (1997, June 13) Electronic Log Books. Owner/Driver p10

OECD (1989) Electronic funds transfer - Plastic cards and the consumer OECD Paris

OKIDATA (1997) OKIDATA comPCard 486-50/100 system overview 3rd Edition

OKIDATA Embedded Products Group New Jersey.

Peterson, I. (1998) The jungles of randomness Penguin Books

Peterson, W.W. (1961) Error-correcting codes John Wiley and Sons

Pfleeger, C.P. (1989) Security in computing Prentice-Hall New Jersey

PRENA Preneel, B., Govaerts, R., & Vanderwalle, J. (1991) Information authentication: hash functions and digital signatures Computer Security and Industrial Cryptography (edited Preneel, B., Govaerts, R., Vanderwalle, J.) Springer-Verlag

PRENB Preneel, B., Govaerts, R., and Vanderwalle, J. (1991) Technical approaches to thwart computer fraud Computer Security and Industrial Cryptography (edited Preneel, B., Govaerts, R., Vanderwalle, J.) Springer-Verlag

Rankl, W. & Effing, W. (1997) Smart Card hand Book John Wiley and Sons

RFC1321 (1992) The MD5 message-digest algorithm Rivest. R.L.

Roberts, S. (1995) Auditable random numbers Proceedings of Cryptology: Policy and Algorithms conference. Queensland University of Technology

Schneier, B. (1996) Applied cryptology 2nd Edition. John Wiley and Sons

Seberry, J. & Pieprzyk, J. (1989) Cryptography: An introduction to computer security Prentice Hall New Jersey.

Sedgewick, R. (1990) Algorithms in C Addison –Wesley

SET6 (1996) Secure electronic transaction (SET) specification (Book 1) Business description Draft Version 17 June 1996

Shaw, D.T. (1993) TeDEUM - Text Decryption Encryption Using Music Final Year Design Project for B. App. Sci Computer and Information Science University of South Australia. The Levels South Aust. 5095 unpublished manuscript.

Sibigtroth, J. (1997, August) When 64KB isn't enough Circuit Cellar INK no. 85 pp18-22.

Smith (1996) Dick Smith Electronics - Catalogue '96 '97 Perth Western Australia

Spiegel, M.R. (1972) Schaum's outline of theory and problems of statistics in SI units 1st edition McGraw Hill.

Strandh, S. (1979) The history of the machine, Bracken Books, England.

TCOD (1977) The Concise Oxford Dictionary Sixth Edition, fourth impression. Edited J.B. Sykes Oxford University Press Oxford.

Vedder, K. (1992) Smart Cards IPO CD-ROM (93-26)

Whyte, D.J. (1997) The development and use of the Secure Electronic Transaction (SET) protocol on the internet Bachelor of Science Honours thesis Edith Cowan University Perth Western Australia

Wolf3D (no date) Wolf 3D Advanced engine management [Brochure] Advanced Engine Management Pty. Ltd. Ringwood Victoria 3134

Young, E.C. (1979) The New Penguin Dictionary of Electronics Editor V. Pitt Penguin Books

Zeng, K., Yang, C.H., Wei, D.Y. & Rao, T.R.N. (1991) Pseudorandom bit generators in stream-cipher cryptography IEE Press

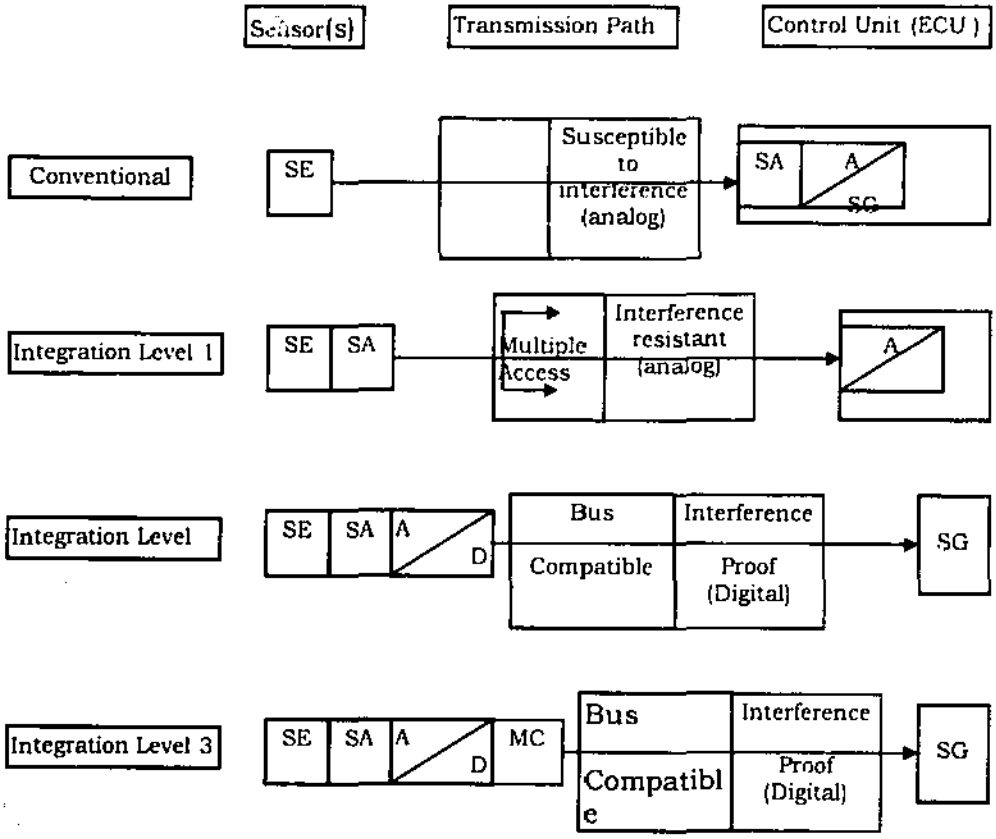
Zoreda, J.L. & Oton, J.M. (1994) Smart Cards Artech House Boston & London

9. Appendices

9.1. Sensor Integration Levels

(BOSCH93 p103)

Legend.
SE - Sensors
SA - Signal Processing (analog)
AD - Analog-digital converter
SG - Control Unit
MC – Microcomputer



9.2. Engine Types and Classifications

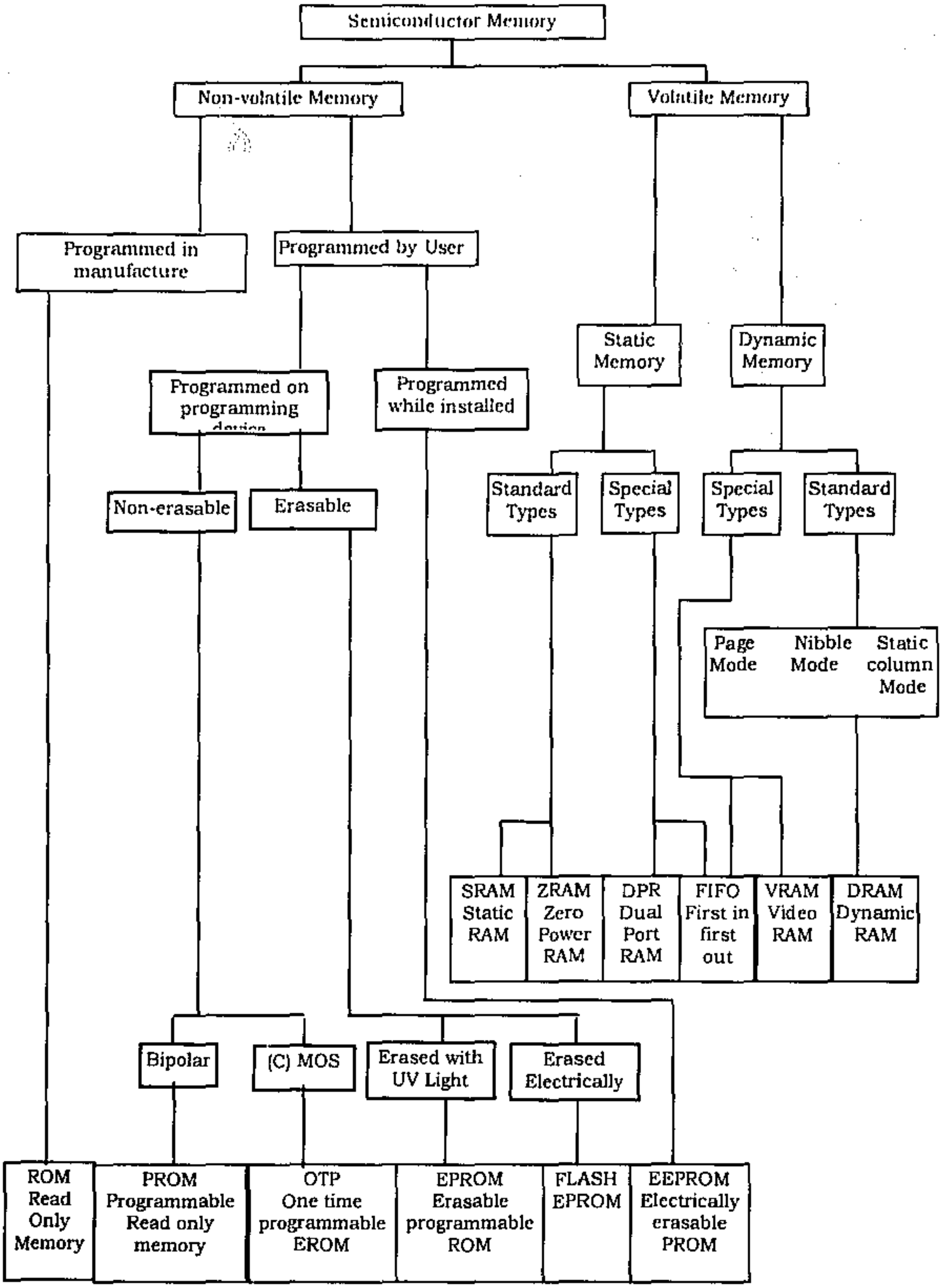
(BOSCH93 p 352)

Type of Process	Open Process				Closed Process	
	Internal Combustion				External Combustion	
	Combustion gas =working medium				Combustion gas ≠ working medium	
					Phase change in working medium	
					No	Yes
Type of Combustion	Cyclic Combustion			Continuous Combustion		
Type of Ignition	Auto-ign	External Ignition				
Enclosed Working Chamber	Diesel	Hybrid	Otto-Cycle	Rohs	Stirling	Steam
Turbine	-	-	-	Gas	Hot Steam	Steam
Type of Mixture	Heterogenous		Homo-geneous	Heterogenous		
	In combustion chamber			In a continuous flame		

Table 9-1 Classification of Engine Types

9.3. Overview of Semiconductor Memory Devices

(Bosch93 p95)



9.4. Information Security Objectives

(Menezes, Van Oorschot and Vanstone p3)

Privacy or Confidentiality	keeping information secret from all but those who are authorised to see it.
Data integrity	ensuring information has not been altered by unauthorised or unknown means
entity authentication or identification	corroboration of the identity of an entity (e.g. a person, a computer terminal, a credit card etc.)
message authentication	corroborating the source of information; also known as data origin authentication
Signature	a means to bind information to an entity
Authorisation	conveyance, to another entity, of official sanction to do or be something
Validation	a means to provide timeliness of authorisation to use or manipulate information or resources.
Access control	restricting access to resources to privileged entities
Certification	endorsement of information by a trusted entity
Timestamping	recording the time of creation or existence of information
Witnessing	verifying the creation or existence of information by an entity other than the creator
receipt	acknowledgment that information has been received
confirmation	acknowledgement that services have been provided
Ownership	a means to provide an entity with the legal right to use or transfer a resource to others.
Anonymity	concealing the identity of an entity involved in some process.
non-repudiation	preventing the denial of previous committments or actions.
Revocation	retraction of certification or authorisation.

Table 9-2 Some information security objectives

9.5. Smart Card Specifications

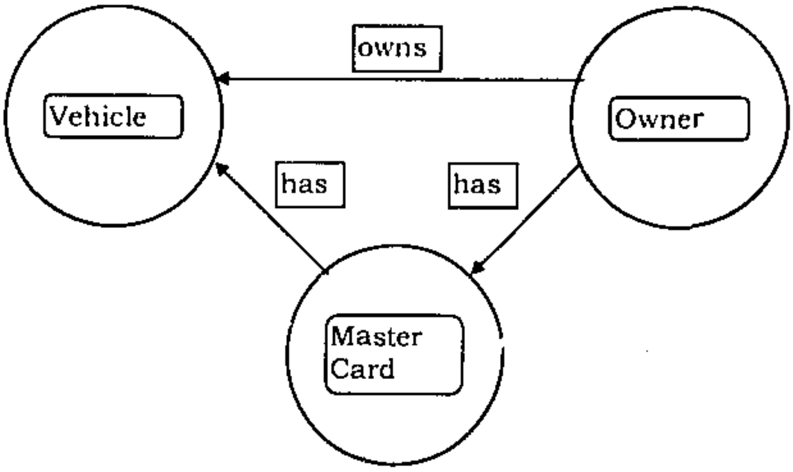
(Hendry p106)

Application	Manufacturer	Processor	ROM	PROM	RAM	EEPROM	Coprocessor	Security Features Included
Phone Card	ODS (Siemens chip)	-	16 bits	56 bits	-	32 bits	-	security logic; countdown only
Eurochip 2nd Generation public telephony	Philips	6805	6 kb	-	128 bytes	1 kb	-	randomised response times; low frequency protected
GSM telephony	SGS-Thomson	6805	16 + 1.5 kb	-	384 bytes	8 kb	Arithmetic	Address scrambling
Bank (credit/debit)	Gemplus	6805	8 kb	-	-	1 - 8 kb	Arithmetic	-
Health / Secure Payments	Solaic	8051	8 + 2 kb	-	256 bytes	2.5 kb	Crypto (RSA)	-
Computer Access	Philips	8051	20 kb		640 bytes	8 kb	Crypto (fast exponentiation)	Low frequency and voltage sensors

Table 9-3 Sample Card Specifications

9.6. Entity Relationship diagrams

VAMP - Top Level Diagram 01.0 Ownership

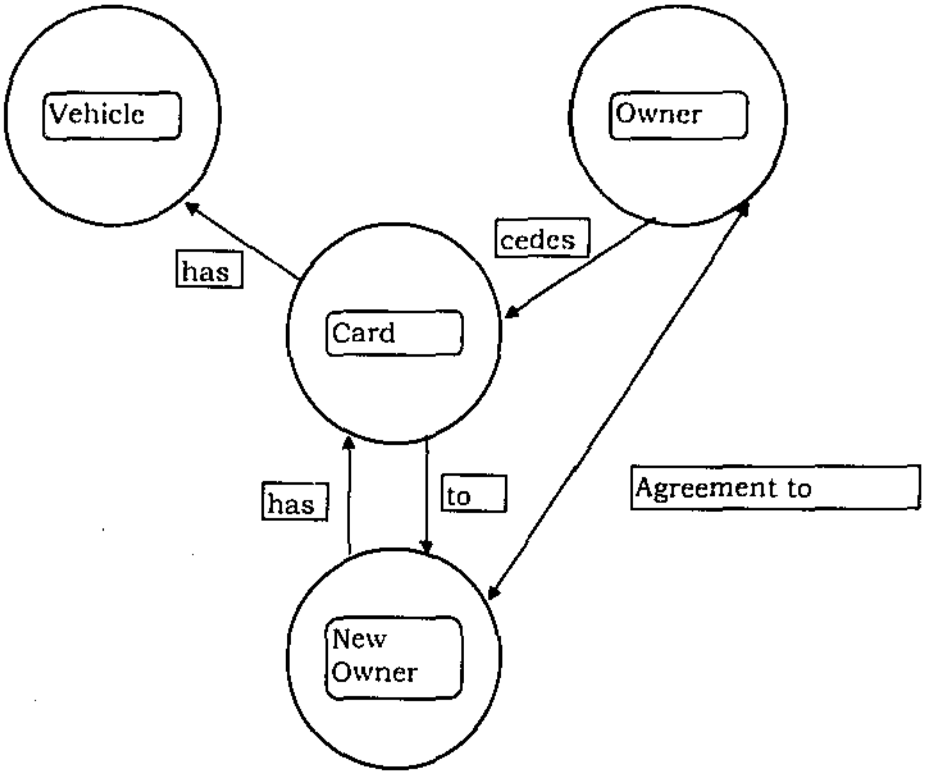


Ownership

1. Owner with legal ownership of Vehicle
2. Owner Has Master Card to Vehicle
3. Master Card has Vehicle

Note. At present, it is assumed that there exists a form of Master Card for the Vehicle which serves as a certificate of ownership and gives complete access to all functions of the control system.

VAMP - Top Level Diagram 01.1 Transfer of Ownership

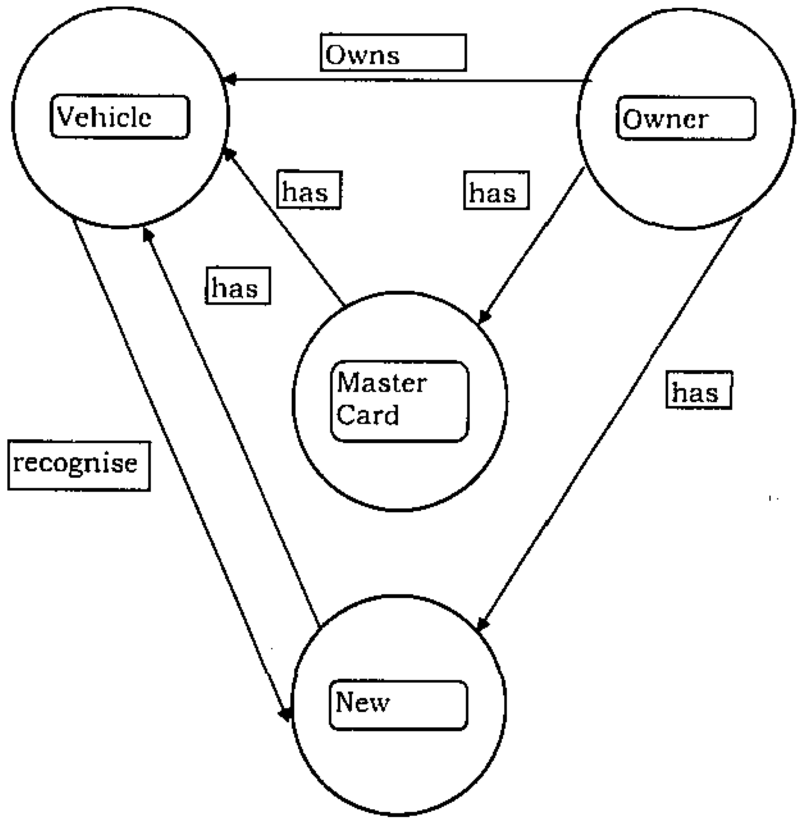


Transfer of Ownership

1. Owner and New Owner reach agreement about sale of Vehicle
2. Owner cedes Master Card to New Owner
3. New Owner has Master Card
4. Master Card has Vehicle

Note. At present, it is assumed that there exists a form of Master Card for the Vehicle which serves as a certificate of ownership and gives complete access to all functions of the control system.

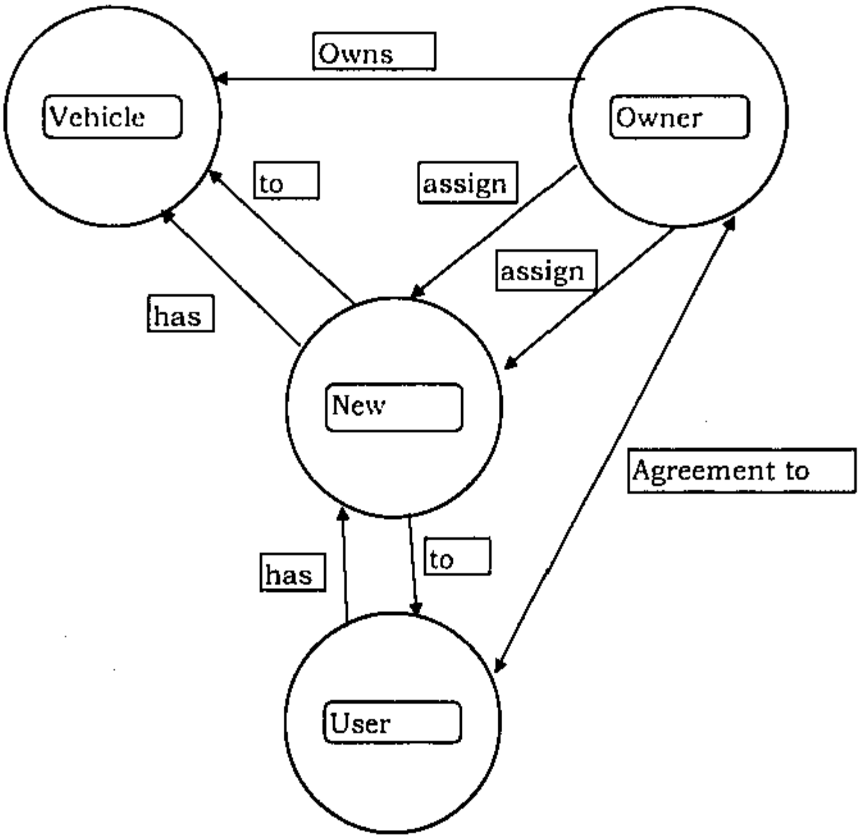
VAMP - Top Level Diagram 01.2 New Card Preparation



Issue of Vehicle

1. Owner with legal ownership of Vehicle
2. Owner uses Master Card to prepare the Vehicle to accept New Card
3. Owner has New Card and allocates it to the car
4. Vehicle recognises New Card
5. New Card has Vehicle

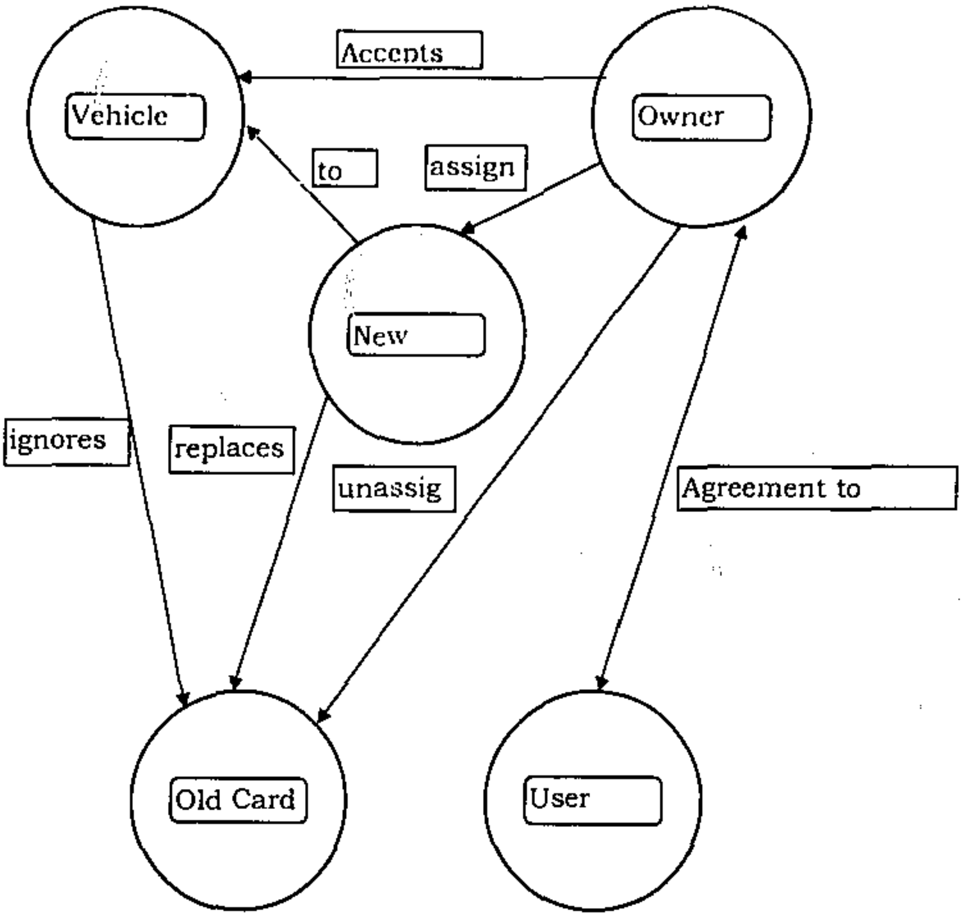
VAMP - Top Level Diagram 02.0 Lease Agreement



Issue of Vehicle

1. Owner with legal ownership of Vehicle
2. Owner assigns New Card to Vehicle (refer 01.2)
3. Owner and User reach agreement about use of Vehicle
4. Owner assigns New Card to User
5. User has New Card
6. New Card has Vehicle

VAMP - Top Level Diagram 02.1 Lease Resolution

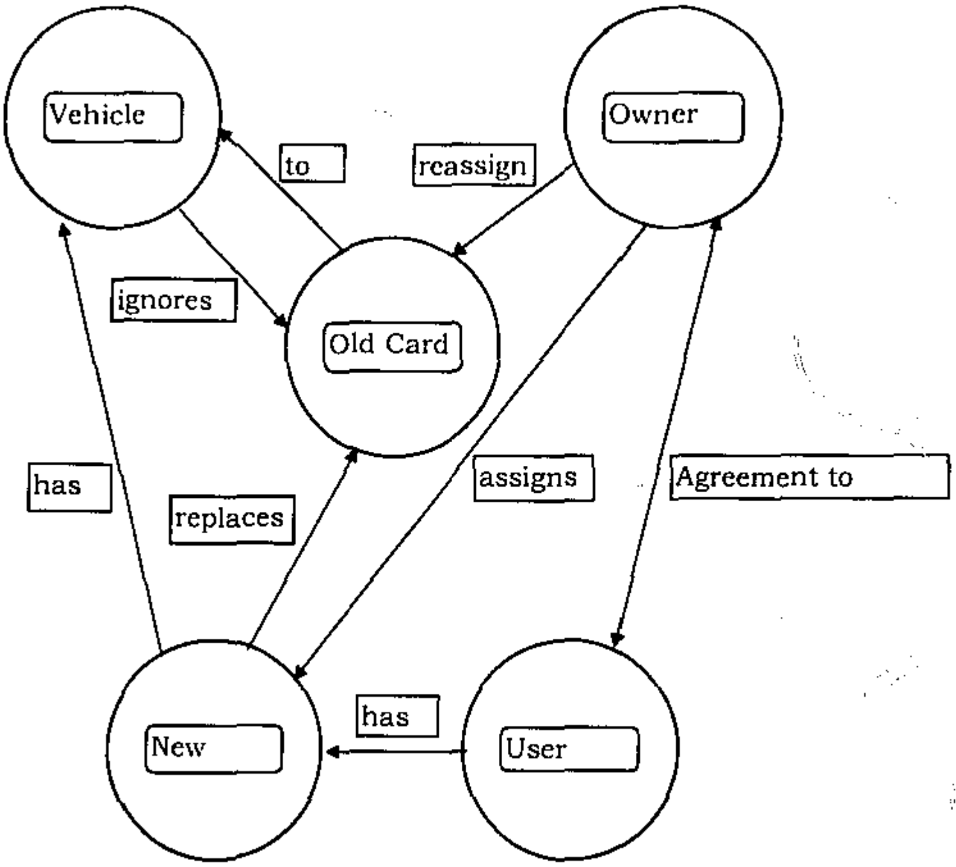


Return of Vehicle

1. Owner and User reach agreement about return of Vehicle
2. Owner accepts Vehicle
3. Owner assigns New Card to Vehicle (refer 01.2)
4. Owner unassigns Old Card from Vehicle
5. New Card replaces Old Card
6. Vehicle ignores Old Card

Note. The Old Card can be retained as a receipt or advertising material and may be destroyed or retained by either owner or user. The assumption is that all Cards are unique and no information is able to be determined about the cards

VAMP - Top Level Diagram 02.2 Lease Extension



Extension of Lease of Vehicle

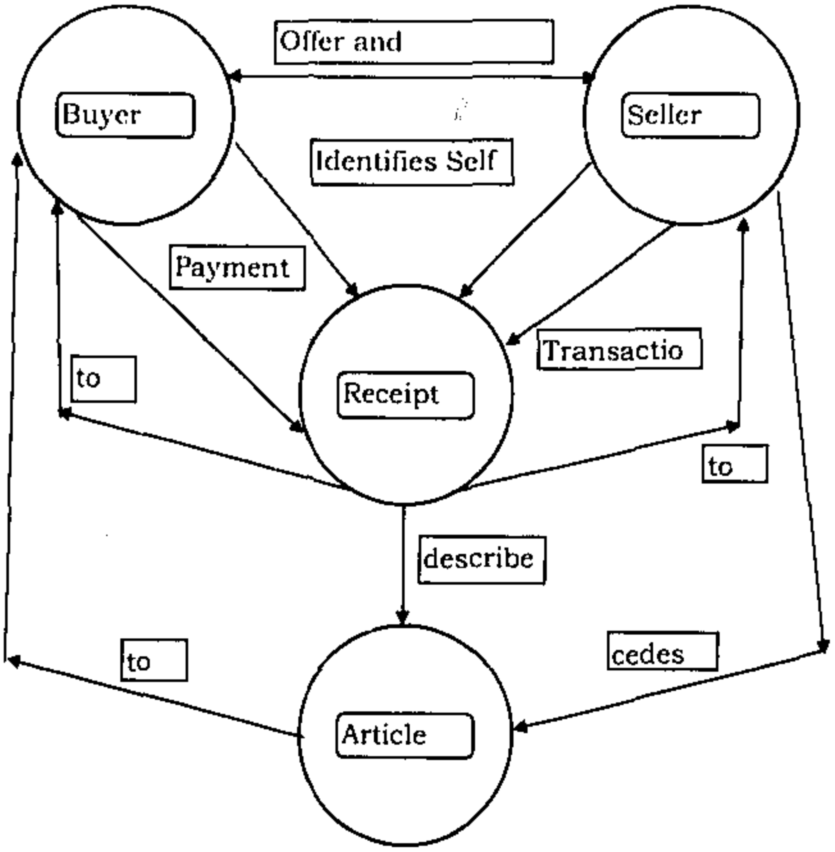
1. Owner and User reach agreement about extension of lease
2. Owner Reassigns Old Card to Vehicle

or

3. Owner assigns New Card to Vehicle (refer 01.2)
4. New Card replaces Old Card
6. User has New Card
7. New Card has Vehicle
8. Vehicle ignores Old Card

Note. The Old Card can be retained as a receipt or advertising and may be destroyed or retained by either owner or user. The assumption is that all Cards are unique and no information is able to be determined about the cards

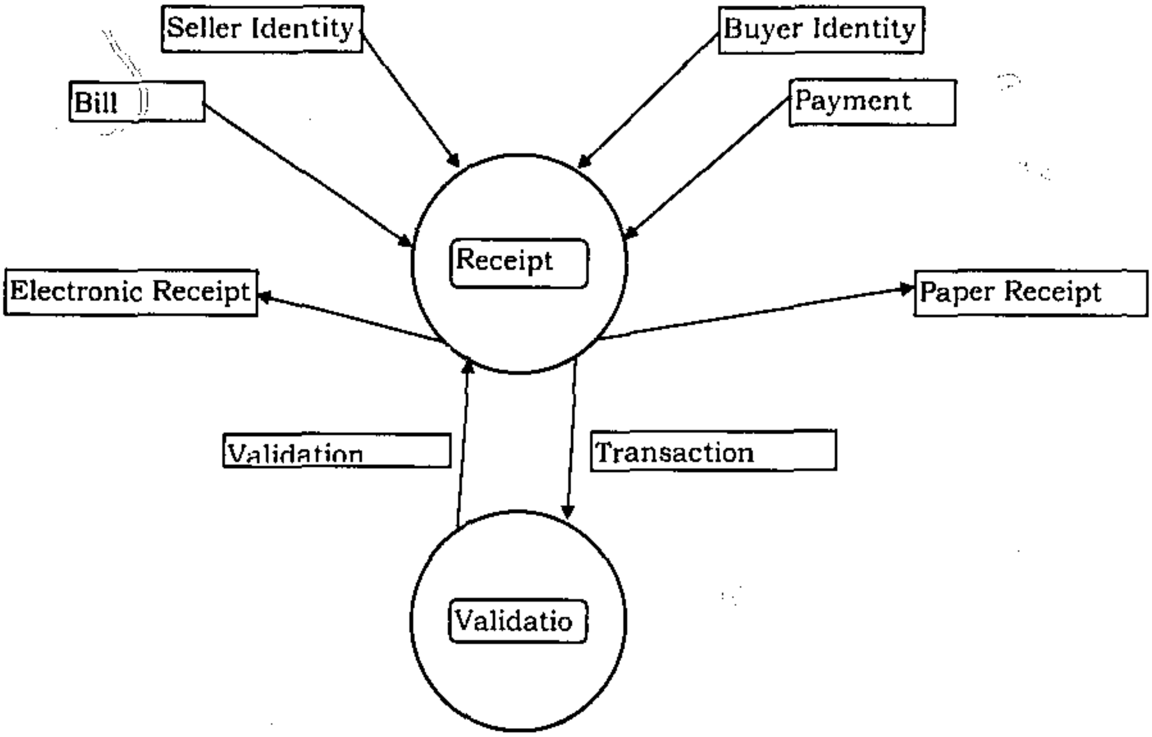
VAMP - Top Level Diagram 03.0 Buying and Selling



Buying and Selling

1. Buyer offers and Seller Accepts Offer
2. Seller Identification on receipt
3. Buyer Identification on receipt
4. Seller provides Transaction details
5. Buyer provides Payment details
6. Electronic copy of Details to Seller
7. Receipt describes Transaction and Article (refer 03.1)
8. Paper copy to Buyer
9. Seller cedes Article to Buyer

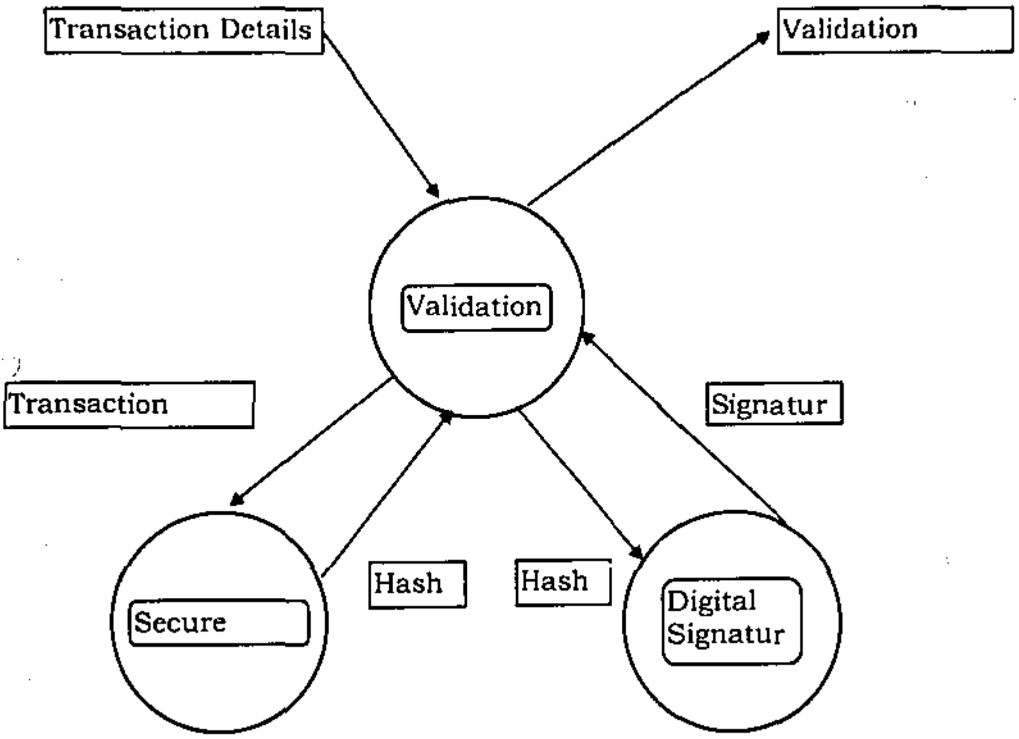
VAMP - Top Level Diagram 03.1 Receipt Processing



Generate a Receipt

1. Seller Identification on Receipt
2. Seller provides Transaction details on Receipt
3. Buyer Identification on Receipt
4. Buyer provides Payment details on Receipt
5. Receipt provides Transaction details to Validation process
6. Validation Process returns Validation details to Receipt
7. Receipt describes Transaction and Article
8. Electronic copy of Details and Receipt to Seller
9. Paper copy of Receipt to Buyer

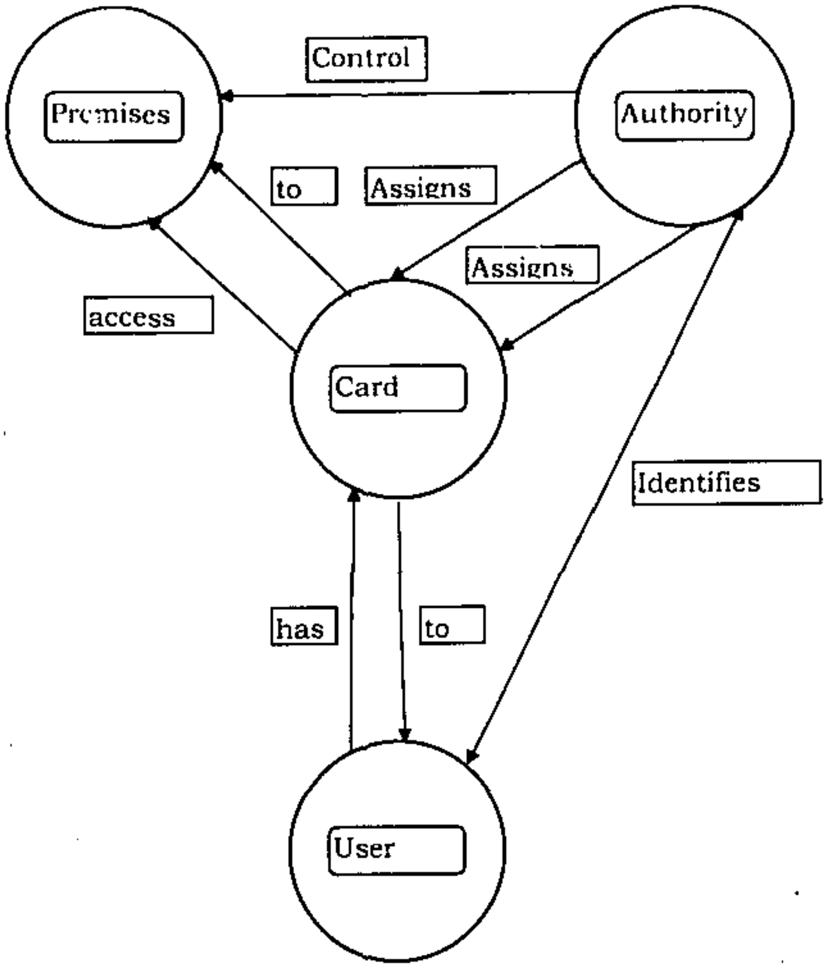
VAMP - Top Level Diagram 03.2 Receipt Processing



Receipt Processing

1. Transaction details from Receipt
2. Transaction details to Secure Hash process
3. Hash result returned
4. Hash to Digital Signature process
5. Digital Signature returned
6. Validation details to Receipt

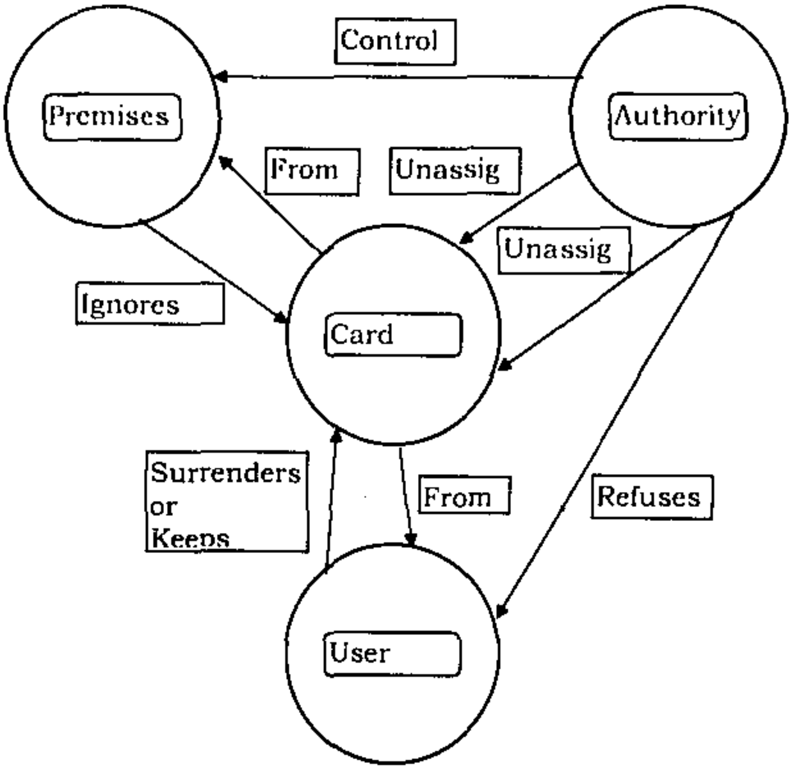
VAMP - Top Level Diagram 04.1 Access Control



Granting Access

1. Authority controls access to premises
2. Authority assigns Card to premises
3. User Identified to Authority
4. Authority assigns Card to User
5. User has Card
6. Card has access to premises

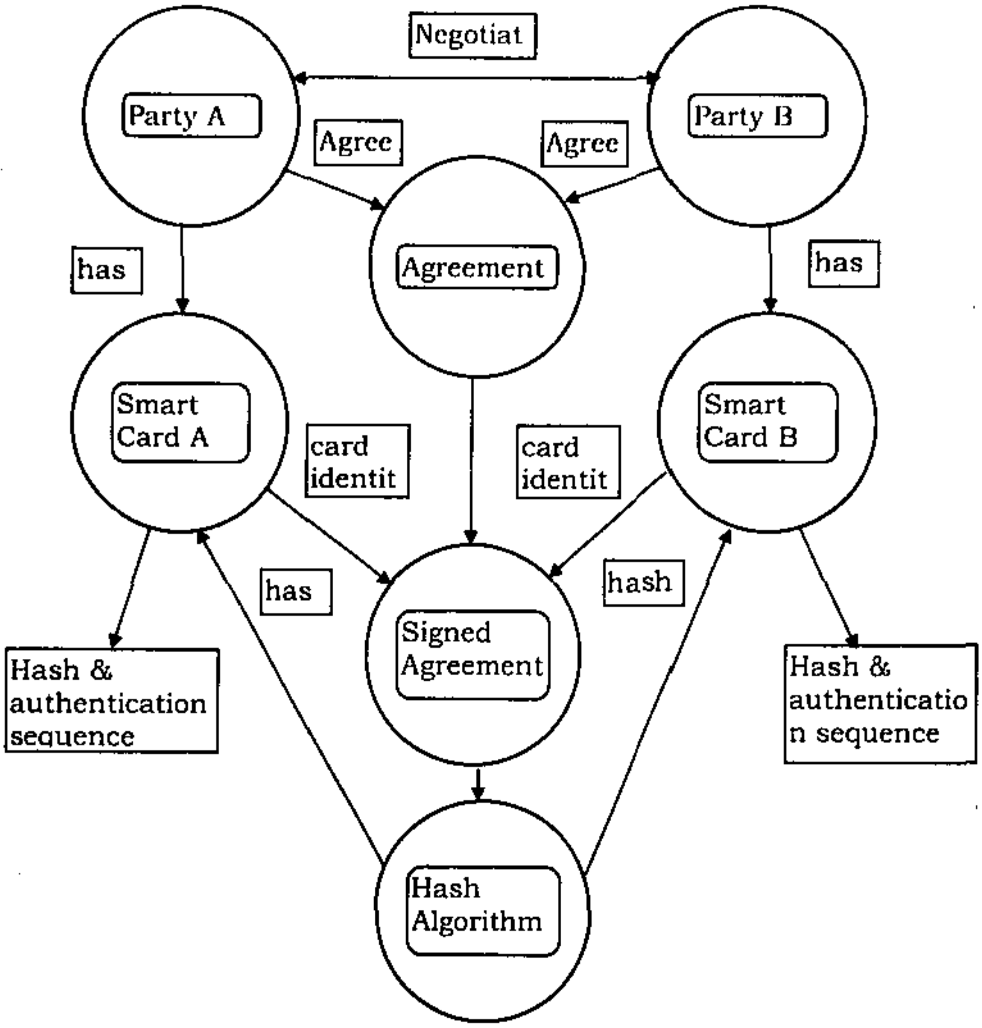
VAMP - Top Level Diagram 04.2 Access Control



<u>Terminates Access</u>

1. Authority controls access to premises
2. Authority refuses User access
3. Authority unassigns Card from User
4. Authority unassigns Card from Premises
5. Premises ignores Card
6. User surrenders or keeps Card

VAMP - Top Level Diagram 05.0 Signature

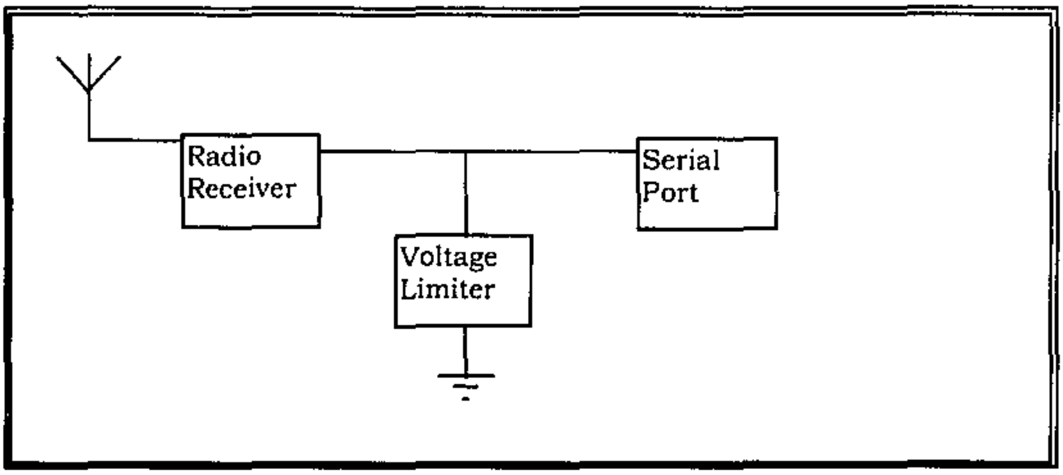


Signature

1. Party A and Party B reach agreement
2. Party A and Party B have unique Smart Cards
3. A and B append a card signature to the agreement
4. The agreement is hashed
5. The hash value is used to generate an Authentication Sequence.
6. The hash and authentication sequences are stored by both parties for reference.

9.7. Random Noise Sampler

This design was provided by Mr. G. Robbins, Research Support Engineer, Edith Cowan University.



9.8. Calculation of Large Factorials and Exponentials

9.8.1. Comparison of Results for Factorial Calculations.

A comparison of results gained by Stirling's Method and the Sequential Summation method shows that, in general, Sequential Summation produces results greater than the results gained with Stirling's Approximation.

	Logarithmic	Stirling
10!	3.6288000000 * 10 ^6	3.5986956250E+00 6
11!	3.9916800000 * 10 ^7	3.9615625127E+00 7
12!	4.7900160000 * 10 ^8	4.7568748747E+00 8
13!	6.2270208002 * 10 ^9	6.1872394893E+00 9
14!	8.7178291203 * 10 ^10	8.6661001955E+00 10
15!	1.3076743680 * 10 ^12	1.3004307256E+00 12
16!	2.0922789888 * 10 ^13	2.0814114472E+00 13
17!	3.5568742811 * 10 ^14	3.5394832971E+00 14
18!	6.4023737057 * 10 ^15	6.3728046458E+00 15
19!	1.2164510041 * 10 ^17	1.2111278698E+00 17
20!	2.4329020081 * 10 ^18	2.4227868550E+00 18
30!	2.6525285982 * 10 ^32	2.6451709723E+00 32
40!	8.1591528323 * 10 ^47	8.1421726985E+00 47
50!	3.0414093197 * 10 ^64	3.0363446196E+00 64
60!	8.3209871117 * 10 ^81	8.3094383964E+00 81
70!	1.1978571669 * 10 ^100	1.1964320185E+00 100
80!	7.1569457061 * 10 ^118	7.1494945686E+00 118
90!	1.4857159644 * 10 ^138	1.4843409665E+00 138
100!	9.3326215494 * 10 ^157	9.3248477868E+00 157
110!	1.5882455416 * 10 ^178	1.5870428136E+00 178
120!	6.6895029130 * 10 ^198	6.6848591871E+00 198
130!	6.4668554867 * 10 ^219	6.4627115428E+00 219
140!	1.3462012447 * 10 ^241	1.3454002075E+00 241

Table 9-4 Comparison of Logarithmic and Stirling's Approximation

It is noted that Stirling underestimates the value of the factorial compared against the logarithmic method and it is suggested that the logarithmic method is a more accurate method of generating large factorials.

9.8.1.1. Calculation by Integer Methods

Comparing Direct Calculation with Sequential Summation shows occasions where the results differ and this is due to the methods of generating logarithms on the computer. Logarithms are approximations of a number where speed of calculation is more important than accuracy. The user must define suitable error levels when using logarithms to calculate large numbers.

It is considered that Integer methods of calculating large factorials are more accurate but substantially slower to use. Above a certain number, the ability to check the accuracy of the Integer method becomes difficult and may be infeasible unless another method becomes available.

	Logarithmic		Integer Calculation
10!	$3.6288000000 * 10^{6}$	=	3628800
11!	$3.9916800000 * 10^{7}$	=	39916800
12!	$4.7900160000 * 10^{8}$	=	479001600
13!	$6.227020800\underline{2} * 10^{9}$	>	6227020800
14!	$8.717829120\underline{3} * 10^{10}$	>	87178291200
15!	$1.3076743680 * 10^{12}$	=	1307674368000
16!	$2.0922789888 * 10^{13}$	=	20922789888000
17!	$3.55687428\underline{11} * 10^{14}$	>	355687428096000
18!	$6.4023737057 * 10^{15}$	<	6402373705728000
19!	$1.216451004\underline{1} * 10^{17}$	>	121645100408832000
20!	$2.4329020081 * 10^{18}$	<	2432902008176640000
30!	$2.652528598\underline{2} * 10^{32}$	>	265252859812191058636308480000000
40!	$8.1591528323 * 10^{47}$	<	815915283247897734345611269596115894272000000000
50!	$3.0414093197 * 10^{64}$	<	30414093201713378043612608166064768844377641568960512000000000000
60!	$8.3209871117 * 10^{81}$	<	8320987112741390144276341183223364380754172606361245952449277696409600000000000000
70!	$1.1978571669 * 10^{100}$	<	11978571669969891796072783721689098736458938142546425857555362864628009582789845319680000000000000000
80!	$7.15694570\underline{61} * 10^{118}$	>	71569457046263802294811533723186532165584657342365752577109445058227039255480148842668944867280814080000000000000000000
90!	$1.4857159644 * 10^{138}$	<	1485715964481761497309522733620825737885569961284688766942216863704985393094065876545992131370884059645617234469978112000000000000000000000
100!	$9.33262154\underline{94} * 10^{157}$	>	93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

Table 9-5 Comparison of Logarithmic and Direct Calculation

The logarithmic result differs from the Integer result after 8 decimal places and this is considered to be due to the methods of generating logarithms. More accurate methods of generating logarithms by direct calculation were not considered even though the algorithm is relatively simple.

In general, logarithms underestimate a number unless the complete logarithm of a number is used in calculation. For example, Pi is irrational and most calculators give Pi as 3.141592654 and the logarithm of Pi to the base 10 as 0.497149872. A truncated value of Pi means that the numbers generated underestimate the correct value.

Logarithmic methods are quicker to implement but limited by the numeric representation of floating point numbers on the computer which appear accurate only to eight decimal places.

9.8.2. Calculation of Large Binary Numbers by Integer Multiplication

Large exponentials are calculated in entirety and listed in the table below.

Each number is displayed in the following form : exponent of 2 ; calculated value

N	2^N
10	1024
20	1048576
30	1073741824
40	1099511627776
50	1125899906842624
60	1152921504606846976
70	1180591620717411303424
80	1208925819614629174706176
90	1237940039285380274899124224
100	1267650600228229401496703205376
200	1606938044258990275541962092341162602522202993782792835301376
500	3273390607896141870013189696827599152216642046043064789483291368096133796404674554883270092325904157150886684127560071009217256545885393053328527589376
1000	10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069376

Table 9-6 Integer Calculation of Powers of Two

Modest values for N produce substantial values of 2^N and the level of accuracy available is considered to be in excess of any reasonable requirements.

It is noted that where n is a multiple of 20 the last two digits of the number are 76 and where n is a multiple of 10 the last two numbers are 24.

9.9. Visual Testing

9.9.1. Single Bit Sequences

The program is called 'LWALK1.PAS' which displays the results of 10 runs of a single bit generator in graphical form. A composite display created from summing the individual values of the runs characterises the output of the randomiser.

9.9.2. Multi-bit Sequences

The Program is called 'DWC.PAS' and includes a library file called 'DWLIB.PAS'. The implementation performs tests up to four bits on a random generator.

The program includes input and output file handling, printing of the results and trace. Input parameters include choice of target shape and size, number of test runs, two, three or four bit tests, overlapping and non-overlapping samples and characteristics calculated from the trace.

9.9.2.1. Parameters.

The program displays the file input menu first to determine if files will be used. After selecting an option from INPUT, OUTPUT, INOUT - (input and output files used) and NONE the program continues with the Data Input screen.

The Data Input Screen permits the user to change the number of runs, the size and shape of the target, the number of bits tested and whether the samples overlap.

Each trace is displayed in blue and then the blue pixels are counted and set to green.

Maximum Runs

User selectable number of runs where the trace starts at the origin and terminates at the periphery.

Run Number

Actual number of the run being performed.

Target Radius

Radius of the circular target and half the dimension of the square target.

Target Area

Area of the target calculated from counting the pixels inside.

Blue Count

The total number of pixels visited during the test

Blue Area

The actual number of pixels illuminated during the test

Blue Density

The result of dividing the Count by the Area

Blue Direction

Not implemented at present

Green Area

The area of the green trace compared with area of the target as a percentage.

Expected Blue

The expected number of bits according to Kaye.

Average Per Cent

calculated from the average blue count divided by expected blue as a percent

Overlap

if set to yes the test is for a binary source with memory.

If set to no the test is for n bit sequences.

Files Input

Displays the file name for data input to the program

Files Output

Displays the file name for data output from the program

Q to Quit

press 'Q' or 'q' to exit the program when this is displayed

P to Print

press 'P' or 'p' to print the screen to the local printer.

Time

System time at time of test

Date

System Date at time of test

9.9.2.2. Testing

In the examples provided, a file called 'TEST1.DAT' was generated and stored with the program using the compiler random function. The file was then used repeatedly as input for a variety of tests and the different tests compared.

9.9.2.3. Results.

The resulting graphs are stored in the appendix 9.13.

9.10. Secure Sequence

```
program sc1;
{
  A program to simulate the input from a secure hashing protocol and
  use it to generate a secure sequence in return.

  author  : David Shaw 0956404
  date    : 08 july 97
  version : 1  initial version. 05jul97.
             2  loop and error flag added. Start_point stored in a
                   file for analysis 08jul97.

  ancillary programs
  sc1_1 : fill a file with the randomizer
  sc1_2 : fill a file with input sequences of integers
  sc1_3 : read the contents of an integer file
}

uses crt;                          { use text mode facilities }

const
  file_error_message         = 'file is incomplete';
  program_error_message  = 'program is terminated due to error';
  continue_message            = 'Press any key to continue';
  terminate_message          = 'This program has terminated correctly';

  sequence_input_file_name = 'd:\programs\rndinp1.dat';
  { name of file of inputs }
  randomizer_file_name = 'd:\programs\rndmise1.dat';
  { name of randomizer file }
  output_file_name = 'd:\programs\rndout1.dat';
  { name of file outputs }
  sp_output_file_name = 'd:\programs\rndsp1.dat';
  { name of file for storing start_points generated }


  no_bits_in  = 128;             { number of bits in input sequence }
  nbi_1 = no_bits_in - 1;

  no_bits_out = 128;             { number of bits in output sequence }
  nbo_1 = no_bits_out - 1;

  size_of_sp_select = 16;        { number of bits used to calculate SP }
  sp_1 = size_of_sp_select - 1;

  size_of_select = 10;           { number of values in select array }
  s_1 = size_of_select - 1;

  size_of_randomizer = 32768;    { 4096 bytes of rom in bits }

  sp_select : array[0..sp_1]       { selection sequence }
              of integer = (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16);
  {
    numbers may be of any value from 0 to nbi_1 and in any order. these
    numbers indicate which bit from the input sequence is used in which
    order to generate a starting point long integer
  }

 selection : array[0..s_1]  { selection sequence }
```

```
              of integer = (1,2,3,4,5,6,7,8,9,10);
  {
    these numbers are the increments added to the starting point and
    may be of any valid integer or long integer value.
    negative numbers may be used
  }

type
  in_sequence  = array[0..nbi_1]  of integer; { input sequence type }
  out_sequence = array[0..nbo_1] of integer;  { output sequence type }
  sp_out_file  = file of longint;             { startpoint file }
  out_file     = file of integer;             { data file type }
  file_name    = string[40];                  { name of file type }

var
  sequence_input_file  : out_file;            { file of inputs }

  randomizer_file      : out_file;            { randomizer file }

  sequence_output_file : out_file;            { file of outputs }

  sp_output_file       : sp_out_file;         { file of start points }

  start_point : longint;           { starting point in randomizer }

  input        : in_sequence;      { input sequence from input file }
  output       : out_sequence;     { output sequence to output file }
  error_flag   : boolean;          { file error flag }
{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}
procedure get_sequence(var f: out_file;
                       var flag : boolean;
                       var i : in_sequence);
{
  get an input sequence from an input file and store the bits in an
array
}

var count : integer;         { loop variable }
begin
  writeln('Getting Input Sequence From File');
  Writeln;

   for count := 0 to nbi_1 do
   begin
     if eof(f) then
     begin
       writeln(file_error_message);
       flag := true;
       exit;
     end; { if eof }
     read(f,i[count]);        { read value and store in array }

   end; { for count }
end; { get_sequence }

procedure generate_SP(i : in_sequence;
                      var sp : longint);
{
  use the input sequence and the constant sp_select to generate a
  starting point.
}
var temp,    { temporary value }
    value,   { incremental value for starting point sp }
    count    { loop variable }
          : longint;
begin
  writeln('generating a starting point from input sequence');
  writeln;
  value := 0;
  for count := 0 to sp_1 do
  begin
    temp := i[sp_select[count]];
    value := value or temp;
    if count < sp_1 then value := value shl 1;
  end; { for count }
  sp := value;
  writeln('start point is ',sp:10);
  writeln;
end; { generate_sp }

procedure generate_output_sequence(var rif : out_file;
                                   var rof : out_file;
                                       sp : longint;
                                   var o : out_sequence);

{
  Use the starting point and the selection sequence to select
  output sequences from the randomizer file and store the output
  sequence in an output file
}
var rifsize,       { random input file size }
    rifpos         { pointer to position in random input file }
            : longint;
    count,         { loop variable }
    temp           { index to selection increments }
            : integer;
begin
  writeln('Generating output sequence');
  writeln;

  reset(rif);
  rifsize := filesize(rif);     { get size of file }
  rifpos := sp;                 { set file position to start point }

  for count := 0 to nbo_1 do
  begin
    if rifpos > rifsize then rifpos := rifpos mod rifsize;
    { ensure file position is always within the file }
    seek(rif,rifpos);           { position pointer and get bit }
    read(rif,o[count]);
    temp := count mod size_of_select;  { calculate next increment }
    rifpos := rifpos + selection[temp];
    write(rof,o[count]);        { write to output file }
  end; { for count }

end; { generate_output_sequence);

{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}
begin  { main }
 clrscr;
```

```
  writeln('Smart Card Simulation Program 1 ');
  { identify and prepare files for use }
  { file of input sequences }
  assign(sequence_input_file, sequence_input_file_name);
  reset(sequence_input_file);

  { file of randomizer data }
  assign(randomizer_file, randomizer_file_name);
  reset(randomizer_file);
  { file of output sequences calculated from input sequence and
    randomizer data }
  assign(sequence_output_file, output_file_name);
  rewrite(sequence_output_file);
  { file of start points generated from the input sequences }
  assign(sp_output_file, sp_output_file_name);
  rewrite(sp_output_file);

  { loop until input file of input sequences is empty }

  error_flag := false;    { set error_flag condition }

  While not EOF(sequence_input_file) do
  begin
    get_sequence(sequence_input_file,error_flag,input);

    if error_flag then
    begin
      writeln(program_error_message);
      exit;  { if true stop program }
    end; { if error_flag }

    generate_SP(input, start_point);
    write(sp_output_file,start_point);
    generate_output_sequence(randomizer_file,sequence_output_file,
                             start_point, output);
  end; { while not eof }

  { close all files before exiting }
  close(sequence_input_file);
  close(randomizer_file);
  close(sequence_output_file);
  close(sp_output_file);
  { finish message }
  Writeln('This program has terminated correctly');
  writeln(continue_message);
  readln;

end.  { main }
```

```
program scl_1;
{
  A program to simulate the input from a secure hashing protocol and
  use it to generate a secure sequence in return.

  This program generates randomizer bits as integers 0 & 1 and stores
  them in a file

  author  : David Shaw 0956404
  date    : 05 july 97
  version : 1  initial version
}

uses crt;                         { use text mode facilities }

const
  no_bits_out = 128;              { number of bits in output sequence }
  size_of_randomizer = 32768;     { 4096 bytes of rom in bits }

type
  out_file = file of integer;
  file_name = string[40];
var
  output_file_name : file_name;
  output_file : out_file;

  value :                  { random value for storage }
        integer;
  count                    { loop counter }
        : longint;


{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}
procedure fill_file(var f : out_file);
begin
  rewrite(f);
  randomize;              { initialize random generator }
  for count := 1 to size_of_randomizer do
  begin
    value := random(2);
    write(output_file, value);
  end;  { for count }
  close(f);
end; { fill_file }

procedure read_file(var f: out_file);
var value : integer;
begin
  reset(f);
  while not eof(f)do
  begin
    read(f,value);
    write(value);
  end; { while not eof }
  close(f);
end; { read_file }


{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++}
```

```
begin  { main }
  clrscr;
  writeln('Smart Card Simulation Program 1_1 ');
  writeln('Random sequence generation ');
  output_file_name := 'd:\programs\rndmisel.dat';
  assign(output_file, output_file_name);

  fill_file(output_file);
  read_file(output_file);

  writeln('File terminated correctly');

end.   { main }
```

```
program sc1_2;
{
  A program to simulate the input from a secure hashing protocol and
  use it to generate a secure sequence in return.

  This program generates sequences of 128 random bits as integers
  0 & 1 and stores them in a file;

  author  : David Shaw 0956404
  date    : 05 july 97
  version : 1 initial version
}

uses crt;                       { use text mode facilities }

const
  no_bits_out = 128;            { number of bits in output sequence }
  no_of_sequences = 100;        { 100 * 128 bits }

type
  out_file = file of integer;
  file_name = string[40];
var
  output_file_name : file_name;
  output_file : out_file;

  value :                { random value for storage }
        integer;
  count                  { loop counter }
        : longint;


{++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}
procedure fill_file(var f : out_file);

var count1, count2, value : integer;

begin
  rewrite(f);
  randomize;            { initialize random generator }
  for count1 := 1 to no_of_sequences do
  begin
    write('*');
    for count2 := 1 to no_bits_out do
    begin
      value := random(2);
      write(f, value);
    end;  { for count2 }

  end;  { for count1 }
  close(f);
end; { fill_file }

procedure read_file(var f: out_file);
var value : integer;
begin
  reset(f);
  while not eof(f)do
 begin
```

```
    read(f,value);
    write(value);
  end; { while not eof }
  close(f);
end; { read_file }


{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}
begin  { main }
  clrscr;
  writeln('Smart Card Simulation Program 1_2');
  writeln('Random sequence generation for selection sets ');
  output_file_name := 'd:\programs\rndinp1.dat';
  assign(output_file, output_file_name);
  { files closed within procedures }
  fill_file(output_file);

  read_file(output_file);
  writeln;
  writeln('press any key to continue');
  readln;
end.   { main }
```

```
program sc1_3;
{
  A program to simulate the input from a secure hashing protocol and
  use it to generate a secure sequence in return.

  This program reads the bits as integers 0 & 1 from a file

  author  : David Shaw 0956404
  date    : 05jul97
  version : 1  initial version
}

uses crt;                       { use text mode facilities }

const
  no_bits_out = 128;            { number of bits in output sequence }
  size_of_randomizer = 32768;   { 4096 bytes of rom in bits }

type
  out_file = file of integer;
  file_name = string[40];
var
  output_file_name : file_name;
  output_file : out_file;

  value :                { random value for storage }
        integer;
  count                  { loop counter }
        : longint;


{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}

procedure read_file(var f: out_file);
var value : integer;
begin
  reset(f);
  while not eof(f)do
  begin
    read(f,value);
    write(value);
  end; { while not eof }
  close(f);
end; { read_file }


{+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
}
begin  { main }
  clrscr;
  writeln('Smart Card Simulation Program 1_1 ');
  writeln('Sequence reading from a file ');
  writeln;
  write('What is file name to be read ? : ');
  readln(output_file_name);
  writeln;

  assign(output_file, output_file_name);
  { file opened and closed in procedure }
  read_file(output_file);
  writeln;
  writeln;
  writeln('This program has terminated correctly. ');

end.   { main }
```

Test Results - Menu and Access Control

Parameter	Input Value	Expected Result	Actual Result	Comment
Input Password	password	access granted	access granted	ok
-No. Of Tries	No.Tries >limit	access denied	access denied	ok
-time out	do nothing	exit session	exit session	ok
Usage Limit	No. <= limit	permit access	permit access	ok
-Excess use	No. > limit	access denied	access denied	ok
Menu choice	0	exit program	exit program	ok
Time Out	Do Nothing	redo password	redo password	ok
-legitimate	1	show S.No.	show S.No.	ok
-legitimate	2	show sequence	show sequence	ok
-legitimate	3	Hash sequence	Hash sequence	ok
-legitimate	4	crypto choice	crypto choice	ok
-legitimate	5	summary	summary	ok
-error	A	ignore it	ignore it	ok
-error	s	ignore it	ignore it	ok
-error	1.3	ignore it	to option 1	?
-error	4232657	ignore it	to option 4	?
-error	-1	ignore it	to option 1	?

9.11. Simulation of Smart Card Design.

9.11.1. Introduction.

The design from chapter four has been implemented with copy of the MD5 hash algorithm from GnomeWorks. The design of the main program and sub-procedures is discussed and testing of the main program is documented.

9.11.2. Design of Program SMARTCD.PAS

9.11.2.1. Constants

Name	Value	Purpose
NumberOfTries	3	login attempt limit then error
Password	123	password for card access
CardSerialNumber	'1234567890123456'	serial number of card
Hexlist	'0123456789ABCDEF'	convert to hexadecimal characters
Delay1	4000	short delay
delay2	8000	medium delay
timeoutlimit	15000	long delay
numberofbitsin	128	size of input strings
numberofbitsout	128	size of output strings
maxbits	255	largest string permitted
maxusage	500	usage limit
size_of_select	10	number of values in select array
size_of_sp_select	16	number of bits used to calculate SP
size_of_randomizer	32768	4096 bytes of rom in bits
nbi_1	numberofbitsin - 1	for 0..n-1 references
nbo_1	numberofbitsout - 1	for 0..n-1 references
sp_1	size_of_sp_select - 1	for 0..15 references
s_1	size_of_select - 1	for 0..n-1 references
window0	1, 1,80,25	full window coordinates
window1	1, 1,80,12	menu window coordinates
window2	1,13,80,25	result window coordinates
window3	10, 5,70, 7	warning window coordinates
pathname	'd:\programs\scard\	path name
randfilename	'rndmize1.dat'	randomiser file name
usagefilename	'usage1.dat'	usage record file name
inputfilename	'input1.dat'	input string
outputfilename	'output.dat'	output string
logfilename	'logfile1.log'	security log
inseqfilename	'inseq1.txt'	input sequence
headermessage	'Smart Card Simulation Software'	standard message
notokmessage	'Access to this card has not been successful'	standard message
errormessage	'Access to this card has been DENIED'	standard message
exitmessage	'Exit from session selected'	standard message
endmessage	'This program has terminated correctly'	standard message
sn_select	1,15,4,13,2,11,10,99, 48,27,66,15,34,33,2, 11	Serial number select values
sp_select	59,2,5,103,123,107,64,68,69,90,70,35,8,33,19,78	start point select values

selection	1,2,3,4,5,6,7,8,9,10	Selection set select values

9.11.2.2. Inputs, Outputs (Top Level Procedures only)

Main Program

choice	:	char	[menu choice]
sDigest	:	string	[hash process string]
session	:	sessionrecord	[global information]

procedure initsession

session	:	Sessionrecord	[global information]	IN OUT

procedure textsetup

i	:	integer	[screen type]	IN
headermessage	:	constant		OUT

procedure getpassword

session	:	Sessionrecord	[global information]	IN OUT
pw	:	integer	[local password]	OUT
tries	:	integer	[no.of access tries]	OUT

procedure closesession

session	:	Sessionrecord	[global information]	IN OUT

9.11.2.3. Algorithms (Top Level Procedures Only)

```
begin  main procedure
      initialise session record
      prepare text screen
      get password
      if OK
            loop while choice is not 0
                  choice 0 clearscreen and display exit message
                  choice 1 display the card and randomiser serial
numbers
                  choice 3 get a text string and hash it with MD5
                  choice 4 get a text string and hash it for a start
point and encrypt it
                  choice 5 display logfile data
                  choice 6  display usage data and bar graph
            endloop choice
      if NOTOK
            display notokmessage
      if ERROR
            destroy randomiser, input,output, log and usage files
            clear screen and display errormessage
      close the session
      display goodbye message
end main program
```

9.12. Testing

9.12.1. Test Results - Menu and Access Control

Parameter	Input Value	Expected Result	Actual Result	Comment
Input Password	password	access granted	access granted	ok
-No. Of Tries	No.Tries >limit	access denied	access denied	ok
-time out	do nothing	exit session	exit session	ok
Usage Limit	No. <= limit	permit access	permit access	ok
-Excess use	No. > limit	access denied	access denied	ok
Menu choice	0	exit program	exit program	ok
Time Out	Do Nothing	redo password	redo password	ok
-legitimate	1	show S.No.	show S.No.	ok
-legitimate	2	show sequence	show sequence	ok
-legitimate	3	Hash sequence	Hash sequence	ok
-legitimate	4	crypto choice	crypto choice	ok
-legitimate	5	summary	summary	ok
-legitimate	6	Usage Record + Card Serial No.	Usage Record + Card Serial No.	ok
-error	A	ignore it	ignore it	ok
-error	s	ignore it	ignore it	ok
-error	1.3	ignore it	to option 1	ok
-error	4232657	ignore it	to option 4	ok
-error	-1	ignore it	to option 1	ok
Serial Number	...	Card Serial No. + Randomiser Serial No.	1234567890123456 + 2404AE08B62BEFA5F3CF94C2386A60A4	ok
Hash Function	this is a test	hash value	0030D220E048D5CEA955633B1C6042A8	ok
	this is aa test	hash value	9536D061304A9B37784E5C34A0404ADE	ok
	this is a test	hash value	0030D220E048D5CEA955633B1C6042A8	ok
Encryption Function	this is a test	Encryption sequence	0030D220E048D5CE	ok
	this is aa test		9536D061304A9B37	ok
	this is a test		0030D220E048D5CE	ok
Usage Record	...	usage record + card serial no.	123/500 :: 1234567890123456 + bar graph	ok

9.12.2. Logfile

```
Input Output function indicator     O::
Input Sequence                30123456789ABCDEF
sequence separator            ::
Output Sequence         FE1BF09FF604AC83

Hash function indicator       H::
Hash Output Sequence          0030D220E048D5CEA955633B1C6042A8

Encryption function indicator E::
Encryption Output sequence    0030D220E048D5CE hash value of input
sequence
```

Sample of Log file

```
O::0123456789ABCDEF :: FE1BF09FF604AC83
H::0030D220E048D5CEA955633B1C6042A8
E::0030D220E048D5CE
O::0123456789ABCDEF :: FE1BF09FF604AC83
H::0030D220E048D5CEA955633B1C6042A8
E::0030D220E048D5CE
E::0030D220E048D5CE
O::0123456789ABCDEF :: FE1BF09FF604AC83
H::0030D220E048D5CEA955633B1C6042A8
E::0030D220E048D5CE
O::0123456789ABCDEF :: FE1BF09FF604AC83
H::0030D220E048D5CEA955633B1C6042A8
E::0030D220E048D5CE
```

9.12.3. Source Code

SmartCD.PAS, a simulation of the design

program smartcd;

Author David Shaw

File Name : SMARTCD.PAS

this program simulates the functions of a design for a smart card proposed in chap 4 of the thesis documents.

version : 1 initial version 15jul97

2 common include file theslib.pas added 05nov97

3 include MD5 unit

4 complete menu options

functions include

1. serial number 16 bytes
2. accept a hash function and return a transfer function
3. generation of a hash function from input data
4. generation of a stream cipher encryption using a hash key
5. storage of transaction details plain or encrypted output
6. Password controlled access
7. function selection

it is considered that a security function that counts the number of unsuccessful attempts to access the card and deletes or destroys the data to prevent any reuse of the card

```
uses crt,             { use text screen calls }
     strings;         { use string processing calls }

{$I d:\programs\scard\sclib.pas}      {include general modules }
{$I d:\programs\scard\md5lib.pas}

var                   { global variables }
  choice   : char;              { menu choice }
  sDigest : string;             { hash process string }
  session  : sessionrecord;    { global information }

{------------------ Main Program ------------------------------------}

begin { smartcd main }

  textsetup(0);              { prepare text screen }
  initsession(session);      { prepare global record }
  getpassword(session);      { validate access request }

  if session.aflag <> error then displayusage(session);

  case session.aflag of      { access request response }

    ok : begin               { password is correct access approved }
           choice := 'X';
      while choice <> '0' do
```

```
           begin
             menu(choice); { check timeout }
             case choice of
{ ok }       '0'  : begin  { exit from session }
                     textsetup(1);
                     gotoxy(22,5);
                     write(exitmessage);
                     delay(delay1);
                    end;

{ ok }       '1'  : getserialnumber(session);  { display card serial no.
}

             '2'  : begin   { output for an input sequence }
                     getinput(session);        { get input string }
                     generateoutput(session);  { display outputstring }
                    end;

{ ok }       '3'  : begin   { hash an input message }
                     gethashinput(session);    { get input string }
                     session.outstring         { perform hash }
                       := MD5ProduceDigest(session.inputstring);
                     putoutsequence(session)   { display outputstring }
                    end;

             '4'  : begin   { encrypt a message }
                     getcryptoinput(session); { get input string }
                     session.tempstring        { perform hash }
                       := MD5ProduceDigest(session.inputstring);
                     encryptmessage(session); { encrypt message }
                    end;

{ ok }       '5'  : displaydata(session);      { display log file }

{ ok }       '6'  : displayusage(session);     { display usage records }
             end; { case choice }
           end; { while choice <> 0 }
         end;  { case ok }
 notok : begin               { access not approved }
           writeln;
           writeln(' ':8,notokmessage);
           delay(delay1);
           {
             resetting of the number of tries is simulated by  closing
             the program. In a real application if 3 tries are made
             before stopping then 24 hours must elapse before the
             numberoftries is set to zero.
           }
         end;
 error : begin               { illegal activity performed or card expired }
           { destroy randomiser and selection algorithms}

           { with session do
             begin
               erase(usagefile);
               erase(inpfile);
               erase(outfile);
               erase(ranfile);
               may not need to
               erase(logfile);
       end; { with }
```

```
                             { display error message }
           textsetup(3);
           textcolor(white + blink);
           textbackground(red);
           clrscr;
           writeln;
           writeln(' ':15,errormessage);
           delay(delay2);

           clrscr;
           textsetup(1);
         end;
 end; { case aflag }
 closesession(session);     { close all files }
 ending;                    { goodbye message }

end. { smartcd main }

Library Program SCLIB.PAS
{
  Author D. Shaw

  SCLIB.PAS is used with SMARTCD.PAS
  include this file to use generic data structures

  used by smartcd.pas
}

const
  numberoftries     =   3;        { login attempt limit then error }
  password          = 123;        { password for card access }

  cardserialnumber = '1234567890123456'; { serial number of card }

  hexlist = '0123456789ABCDEF'; { convert to hexadecimal characters }

  { time constants}
  delay1          =  4000;        { delay periods }
  delay2          =  8000;
  timeoutlimit    = 15000;

  { numeric constants }
  numberofbitsin  = 128;          { size of input strings }
  numberofbitsout = 128;          { size of output strings }
  maxbits         = 255;          { largest string permitted }
  maxusage        = 500;          { usage limit }

  size_of_select     =    10;   { number of values in select array }
  size_of_sp_select  =    16;   { number of bits used to calculate SP }
  size_of_randomizer = 32768;   { 4096 bytes of rom in bits }


  { derived numeric constants }
  nbi_1 = numberofbitsin    - 1; { for 0..n-1 references }
  nbo_1 = numberofbitsout   - 1; { for 0..n-1 references }
  sp_1  = size_of_sp_select - 1; { for 0..15 references  }
  s_1   = size_of_select    - 1; { for 0..n-1 references }

 { text windows }
```

```
  window0 : array[1..4] of integer = ( 1, 1,80,25);  { full }
  window1 : array[1..4] of integer = ( 1, 1,80,12);  { menu }
  window2 : array[1..4] of integer = ( 1,13,80,25);  { result }
  window3 : array[1..4] of integer = (10, 5,70, 7);  { warning }

  { file names }
  pathname          = 'd:\programs\scard\';   { path name }
  randfilename      = 'rndmize1.dat';         { randomiser }
  usagefilename     = 'usage1.dat';           { usage record }
  inputfilename     = 'input1.dat';           { input string }
  outputfilename    = 'output.dat';           { output string }
  logfilename       = 'logfile1.log';         { security log }
  inseqfilename     = 'inseq1.txt';           { input seqeunce }

  { text messages }
  headermessage = 'Smart Card Simulation Software';
  notokmessage  = 'Access to this card has not been successful';
  errormessage  = 'Access to this card has been DENIED';
  exitmessage   = 'Exit from session selected';
  endmessage    = 'This program has terminated correctly';

  { constant arrays }
  sn_select : array[0..15]  { serial number selection sequence }
      of integer = (1,15,4,13,2,11,10,99,48,27,66,15,34,33,2,11);
  {
    numbers may be of any value from 0 to nbi_1 and in any order. these
    numbers indicate which bit from the input sequence is used in which
    order to generate a serial number
  }

  sp_select : array[0..sp_1]       { start  point selection sequence }
              of integer =
(59,2,5,103,123,107,64,68,69,90,70,35,8,33,19,78);
  {
    numbers may be of any value from 0 to nbi_1 and in any order. these
    numbers indicate which bit from the input sequence is used in which
    order to generate a starting point long integer
  }

  selection : array[0..s_1]  { incremental selection sequence }
              of integer = (1,2,3,4,5,6,7,8,9,10);
  {
    these numbers are the increments added to the starting point and
    may be of any valid integer or long integer value.
    negative numbers may be used
  }


type
  { enumeration types }
  accessflag = (ok,notok, error);   { access control flag }
  messagetype = (input,output);     { message buffer index }

  { string types }
  shortstring  = string[16];        { 16 byte message }
  hexstring    = string[32];        { hexadecimal string }
  file_name    = string[40];        { name of file type }
  anystring    = string[80];        { generic string }
  longstring   = string[255];       { largest string permitted }

 { array types }
```

```
  in_sequence  = array[0..nbi_1]    of shortint; { input sequence type }
  intarray     = array[0..maxbits] of shortint;
  inputarray   = array[1..numberofbitsin] of shortint;


  { record types }
  usagerecord  = record           { current state of card usage }
    psword      : integer;        { password for access control }
    usecounter  : integer;        { current usage counter }
    numbertries : integer;        { number of access tries permitted }
  end;  { usagerecord }

  { file types }
  usagefile    = file of usagerecord;  { file of usage record }
  sintfile     = file of shortint;     { short integer file }
  log_file     = text;

  { record type }
  sessionrecord =                 { current state of session }
     record
       serialnumber : shortstring; { card serial number }
       aflag        : accessflag;  { access permitted }

       usage        : usagerecord; { current status of program }

       usagefile    : usagefile;   { file of session record }

       inpfile,                    { input file of characters }
       outfile,                    { output file of shortint }
       ranfile      : sintfile;    { randomiser file of shortint }

       inseqfile,                  { input sequence to be transformed }
       logfile     : text;         { log data file }


       inarray      : inputarray;  { array of shortint }

       inputstring  : longstring;  { input string for hash 255 chars }

       tempstring,                 { work space strings }
       instring     : shortstring;

       outstring,                  { hexadecimal strings }
       logstring    : hexstring; { logfile entry }
     end;  { sessionrecord }


{---------------------- functions --------------------}
function getbit(v,b : integer): integer;
var temp : integer;
begin          { extract a bit from a byte }
  temp := 0;
  case b of
  0 : temp := v and   1;
  1 : temp := v and   2;
  2 : temp := v and   4;
  3 : temp := v and   8;
  4 : temp := v and  16;
  5 : temp := v and  32;
  6 : temp := v and  64;
  7 : temp := v and 128;
  end; { case }
  getbit := temp;
end; { function getbit }

{------------------ procedures ------------------------------}

procedure textsetup(i : integer);
begin      { set up text window and colours }
  case i of                { set text window size and location }
    0 : window(window0[1],window0[2],window0[3],window0[4]);
    1 : window(window1[1],window1[2],window1[3],window1[4]);
    2 : window(window2[1],window2[2],window2[3],window2[4]);
    3 : window(window3[1],window3[2],window3[3],window3[4]);
  end; { case }

  textcolor(yellow);       { default text colour }
  textbackground(blue);    { default background colour }
  clrscr;                  { reset screen }
  gotoxy(20,1);
  writeln(headermessage); { display message }
end; { textsetup }

procedure initsession(var s : sessionrecord);
begin      { preset session paramters }
  with s do
  begin
    serialnumber := cardserialnumber; { set password }
    aflag := notok;                   { default no access }
          { preset files }
    assign(usagefile, pathname + usagefilename);
    assign(inpfile,   pathname + inputfilename);
    assign(outfile,   pathname + outputfilename);
    assign(ranfile,   pathname + randfilename);
    assign(logfile,   pathname + logfilename);
    assign(inseqfile, pathname + inseqfilename);

    reset(usagefile);                 { get current usage state }
    read(usagefile,usage);
    close(usagefile);

    with usage do { put usage data into current buffer }
    begin
      if usecounter > maxusage then { check if card expired }
         s.aflag := notok;

      numbertries := numberoftries; { max permitted tries before lockout
}
      psword      := password;      { store password }
      inc(usecounter);              { increment usage count }
    end; { with usage }
  end; { sessionrecord }
end; { initsession }

procedure getpassword(var s : sessionrecord);
{
  obtain a password from the user and compare it against
  the stored value.  there is no facility for the password
  to be changed.
}
var pw    : integer;     { get integer password / pin number  }
    tries : integer;     { number of tries at access }
```

```
begin
  s.aflag  := notok;   { initialise }
  tries := 0;
  with s.usage do
  begin
    while tries <= numbertries  do   { not excessive tries }
    begin
      textsetup(1);                       { prepare screen }
      gotoxy(10,4);
      write('Input the password now : '); { get password }
      read(pw);
      tries := tries + 1;                 { increment try counter }

      if pw = psword then                 { password entered ok }
      begin
        s.aflag := ok;                    { set ok condition }
        exit;                             { exit while loop }
      end;

      if pw = 0 then                      { user chooses to stop }
      begin
        s.aflag := notok;                 { set notok condition }
        exit;                             { exit while loop }
      end; { if pw }
      { too many tries at guessing password }
      if tries > numbertries then s.aflag := error;
    end; { while tries }
  end; { with u }
end; { getpassword }

procedure Menu(var c : char);
var timeout : integer; { current time }
begin     { display choices and response }
  textsetup(1);      { initiailise text screen }
  timeout := 0;      { clear timer }
  gotoxy(20,1);
  write(headermessage);
  gotoxy(10,4);      { display menu }
  write('0 : Exit from Session');
  gotoxy(10,5);
  write('1 : Display serial number of card');
  gotoxy(10,6);
  write('2 : Generate Output Sequence from Input Sequence');
  gotoxy(10,7);
  write('3 : Generate a hash value for an input Message');
  gotoxy(10,8);
  write('4 : Encrypt a message with a Hash value');
  gotoxy(10,9);
  write('5 : Display Log File');
  gotoxy(10,10);
  write('6 : Display Usage Record');

  gotoxy(20,12);
  write('Enter Choice : ');

  c := '0';          { default response = exit }
  repeat             { input process loop }
      inc(timeout);  { increment usage time }
      if keypressed then
      begin
        c := readkey;   { calculate input value }
        writeln(c);     { echo to screen }
        exit;
      end;
  until timeout >= timeoutlimit;

  write('Access Timeout');  { if timeout and no keypressed then exit }
  delay(delay1);
  clrscr;
end;  { menu }

procedure closesession(var s : sessionrecord);
begin                { close current session }
  with s do
  begin
    aflag := notok;        { disallow access }
    rewrite(usagefile);    { store usage data at end of session }
    write(usagefile,usage);
    close(usagefile);
  end; { with }
end; { closesession }

procedure ending;
begin                      { display goodbye message }
  textsetup(3);            { initialise screen }
  gotoxy(10,3);
  writeln(endmessage);
  delay(delay1);
end; { ending }

{----------------- option 1 procedures-------------------------------}

procedure getserialnumber(s :  sessionrecord);

var       { generate randomiser serial number }
    bits,              { loop  counters }
    chars : integer;   { characters in sequence }
    ch    : byte;      { hex character equivalent }
    lint  : shortint;  { input bit from randomiser }
begin
  textsetup(2);          { initialise text screen }

  reset(s.ranfile);      { open file }
  ch := 0;
  gotoxy(5,3);
  write('SERIAL : ');
  write(s.serialnumber); { write card serial number }
  write(' :: ');

  for chars := 1 to 32 do  { get first 128 bits from file }
  begin
    for bits := 1 to 4 do
    begin
      read(s.ranfile,lint);
      ch := ch shl 1;      { shift left 1 place }
      ch := ch or lint;    { add LSB ch or 0000000X}
      ch := ch and 15;     { blank 4 msb digits ch and 0000XXXX}
    end;  { for bits }
    write(copy(hexlist,ch+1,1));    { convert to hexadecimal chars }
    ch := 0;
  end;  { for chars }
```

```
9.12.4.     sp := sp + getbit(intvalue,bit);  { get bit }
      sp := sp shl 1;                        { make sp }
    end; { for count }
end;  { startpoint }

procedure generateoutput(var s : sessionrecord);
{
  open an input file and generate a sequence according to
  a particular startpoint and selection algorithm
}
var          { convert input string to bits }
  linarray       : in_sequence; { input string for conversion }
  loutarray      : in_sequence; { output sequence from conversion }
  index,                        { index to selection set }
  bits           : integer;     { loop counter }
  filelength,                   { size of randomiser }
  start          : longint;     { position in randomiser }
  logstring      : shortstring; { log file record }
begin
  textsetup(1);                        { initialise screen }
  startpoint(start,s.instring);        { generate a start point }

  reset(s.ranfile);                    { generate output sequence }
  filelength := filesize(s.ranfile);   { get length of randomiser }
  start := start mod filelength;       { calculate start point }

  for bits := 0 to nbi_1 do
  begin
    seek(s.ranfile,start);             { seek position in randomiser }
    read(s.ranfile,loutarray[bits]);   { read bit }
    index := bits mod 16;              { get next bit index }
    start := start + sn_select[index]; { increment selection value }
    start := start mod filelength;     { make sure it is inside file }
  end; { for bits }

  close(s.ranfile);

  append(s.logfile);                   { store details in log file }
  textsetup(2);                        { initialise screen }
  gotoxy(5,3);
  write('The input sequence is  : ',s.instring);
  write(s.logfile,'O::',s.instring);
  gotoxy(5,5);

  write('The output sequence is : ');
  convert2hex(logstring,loutarray);
  writeln(logstring);
  writeln(s.logfile,' :: ', logstring);
  close(s.logfile);

  delay(delay1);
  clrscr;
  textsetup(1);
end; { generateoutput }

{----------------- option 3 procedures-----------------------------}

procedure gethashinput(var s : sessionrecord);

begin         { get a string to hash }
  textsetup(1);       { initialise screen }
  gotoxy(5,5);
  write('HASH : What is the input message ? : ');
  readln(s.inputstring);

  if length(s.inputstring) = 0 then { clear buffer }
    read(s.inputstring);
  writeln;
end;  { gethashinput }

procedure putoutsequence(var s : sessionrecord);
begin          { display output sequence }
  gotoxy(5,10);
  write('The output sequence is : ');
  write(s.outstring);
  append(s.logfile);    { write hash value to logfile }
  writeln(s.logfile,'H::',s.outstring);
  close(s.logfile);
  delay(delay1);
end; { putoutsequence }

{------------------ option 5 procedures-------------------------------}

procedure getcryptoinput (var s : sessionrecord);
begin                    { get input string }
  textsetup(1);          { initialise screen }
  gotoxy(5,5);
  write('CRYPTO : ');
  readln(s.inputstring); { get input string }

  if length(s.inputstring) = 0 then  { clear buffer }
     readln(s.inputstring);          { get input string }
end;  { getcryptoinput }

procedure encryptmessage(var s : sessionrecord);
var
    msg,                     { message bit }
    key,                     { key stream bit }
    bit      : shortint;     { process bit process }
    filelength,              { randomiser size }
    sp       : longint;      { start point }
    index,                   { index to array }
    loop,                    { loop counter }
    count,                   { loop coounter }
    inlength : integer;      { length on string }
    ch       : byte;         { character in string }
begin
  textsetup(1);              { initialise }
  gotoxy(5,5);
  rewrite(s.inpfile);
  rewrite(s.outfile);
  reset(s.ranfile);
  append(s.logfile);
  index := 0;
  startpoint(sp,s.tempstring);  { calculate a startpoint }

  { convert inputstring to input file of shortints }

  inlength := length(s.inputstring);  { number of characters to process
}

  for count := 1 to inlength do
  begin
    ch := ord(s.inputstring[count]);   { get a character }
    for loop := 1 to 8 do   { convert to bits and store }
    begin
      bit := ch and 1;
      ch := ch shr 1;
      write(s.inpfile,bit);
    end; { for loop }
  end; { for count }

  reset(s.inpfile);
  filelength := filesize(s.ranfile); { get length of randomiser }
  sp := sp mod filelength;           { set start point }
  seek(s.ranfile,sp);
  while not EOF(s.inpfile) do        { process message bits }
  begin
    inc(index);                      { selector for array }
    read(s.ranfile,key);             { get bits from files }
    read(s.inpfile,msg);

    sp := sp + sp_select[index mod 16]; { increment selector }
    sp := sp mod filelength;         { ensure less than randomiser size
}
    if key = msg then                { if same then 0 }
      bit := 0
    else                             { if different then 1 }
      bit := 1;
    write(s.outfile,bit);            { store crypto bit }
  end;  { while }

  gotoxy(5,10);
  write('OUTPUT : ');                { reset crypto file }
  reset(s.outfile);
  while not eof(s.outfile) do        { convert file to ascii }
  begin
    ch := 0;
    for count := 1 to 8 do           { convert char to ascii }
    begin
      read(s.outfile,bit);
      ch := ch shl 1;
      ch := ch or bit;
    end; { for }
    write( chr(ch));                 { write ascii char }
  end; { while not eof }
  writeln(s.logfile,'E::',s.tempstring); { store tempstring }

  close(s.inpfile);                      { close files }
  close(s.outfile);
  close(s.ranfile);
  close(s.logfile);

  delay(delay1);
end; { encryptmessage }

{------------------ option 5 procedures------------------------------}

procedure displaydata(var s: sessionrecord);
var               { display logfile data }
    linecount     : integer;     { number of lines displayed }
    inputsequence : anystring;   { input string from file }
begin
```

```
  reset(s.logfile);       { open logfile }
  linecount := 0;
  textsetup(1);           { initialise screen }
  gotoxy(5,5);
  writeln('LOG : ');
  while not eof(s.logfile) do
  begin
    inc(linecount);                   { count line }
    readln(s.logfile,inputsequence);  { get line }
    writeln(inputsequence);           { display line }
    if linecount mod 10 = 0 then      { pause to read lines }
    begin
      writeln('Enter to continue ');
      readln;
    end; { if }
  end; { while }
  write('FINISH : ');                 { close files }
  close(s.logfile);
  delay(delay1);
end; { displaydata }

{---------------- option 6 procedures ----------------------}

procedure displayusage(var s : sessionrecord);
var maxuse,  { maximum usage in tens }
    tenuse,  { actual usage in tens }
    oneuse,  { actual usage in ones }
    count : integer;  { loop counter }
begin                  { display current usage status }
  clrscr;
  textsetup(2);        { initialise screen }

  maxuse := maxusage div 10; { max uses of card in 10's }
  tenuse := s.usage.usecounter div 10; { use by tens }
  oneuse := s.usage.usecounter mod 10; { use by units }

  gotoxy(10,4);       { display usage }
  write('USAGE : ',s.usage.usecounter:5,'/',maxusage);
  writeln('  Serial : ',s.serialnumber);  { display card serial number }

  { display bar graph }
  gotoxy(10,5);  for count := 1 to maxuse do write('°');
  Gotoxy(10,5);  for count := 1 to tenuse do write('²');

  gotoxy(10,6);  for count := 1 to 9 do write('°');
  gotoxy(10,6);  for count := 1 to oneuse do write('²');

  { if maximum usage limit exceeded destroy card }
  if s.usage.usecounter > maxusage then s.aflag := error;
  textsetup(1);
end; { displayusage}
```

4.3 MD5LIB.PAS MD5 Hash Program Library

```
{ program md5lib .pas ;}

{
this program is taken from the network. Gnomeworks 1998  modified to run
under turbo pascal but not perfect  modified to work with the Smart Card
Simulation Program
}
```

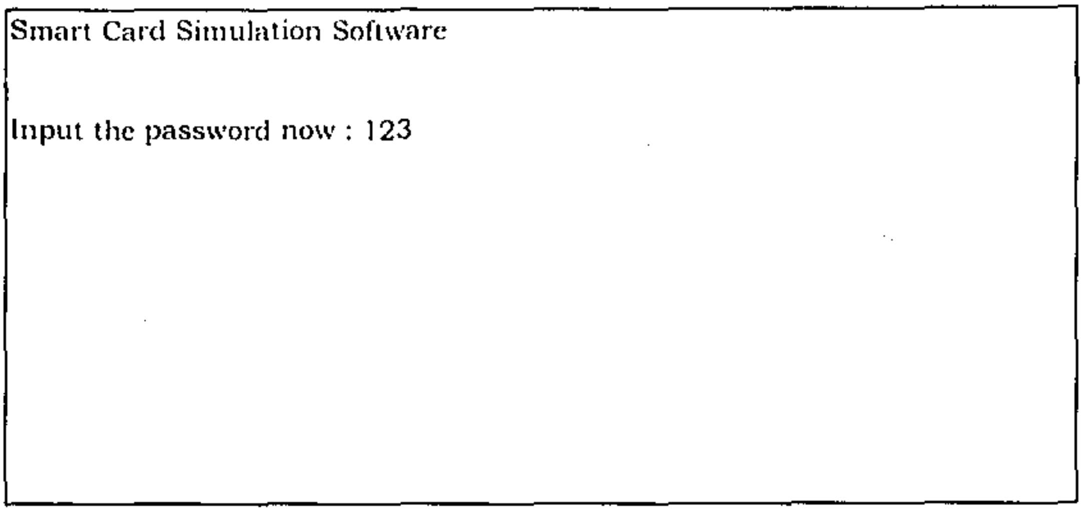
```
const
  hexlisting = '0123456789ABCDEF';


  S11 = 7;
  S12 = 12;
  S13 = 17;
  S14 = 22;
  S21 = 5;
  S22 = 9;
  S23 = 14;
  S24 = 20;
  S31 = 4;
  S32 = 11;
  S33 = 16;
  S34 = 23;
  S41 = 6;
  S42 = 10;
  S43 = 15;
  S44 = 21;

type
  MD5_CTX = record
     state : Array[0..4] of LongInt;
     count : Array[0..2] of LongInt;
     buffer : Array[0..64] of Char;
  end;

  PMD5_CTX = ^MD5_CTX;
  PLInt = ^LongInt;

var
  PADDING : Array[0..63] of Char;

{-------------------------------------------------------}
procedure RotateLeft(var x : LongInt; n : Integer);
var
  Result1 : LongInt;
  xTemp : LongInt;
begin
  xTemp := x;
  Result1 := (xTemp shl n) or (xTemp shr (32-n));
  x := Result1;
end;


function F(x : LongInt; y : LongInt; z : LongInt) : LongInt;
var
  result1 : LongInt;
  result2 : LongInt;
begin
  result1 := x and y;
  result2 := (not x) and z;
  F       := result1 or result2;
end;

function G(x : LongInt; y : LongInt; z : LongInt) : LongInt;
var
  result1 : LongInt;
  result2 : LongInt;
```

```
begin
  result1 := (x and z);
  result2 := y and (not z);
  G       := result1 or result2;
end;

function H(x : LongInt; y : LongInt; z : LongInt) : LongInt;
begin
  H := (x xor y xor z);
end;

function I(x : LongInt; y : LongInt; z : LongInt) : LongInt;
var
  result1 : LongInt;
begin
  result1 := x or (not z);
  I := y xor result1;
end;

procedure FF(var a : LongInt; b,c,d : LongInt; x : LongInt; s : LongInt
; ac : LongInt);
begin
  a := a + F(b,c,d) + x + ac;
  RotateLeft(a,s);
  a := a + b;
end;

procedure GG(var a : LongInt; b,c,d : LongInt; x : LongInt; s : LongInt
; ac : LongInt);
begin
  a := a + G(b,c,d) + x + ac;
  RotateLeft(a,s);
  a := a + b;
end;

procedure HH(var a : LongInt; b,c,d : LongInt; x : LongInt; s : LongInt
; ac : LongInt);
begin
  a := a + H(b,c,d) + x + ac;
  RotateLeft(a,s);
  a := a + b;
end;

procedure II(var a : LongInt; b,c,d : LongInt; x : LongInt; s : LongInt
; ac : LongInt);
begin
  a := a + I(b,c,d) + x + ac;
  RotateLeft(a,s);
  a := a + b;
end;


procedure MD5_memcpy(output : PChar ; input : PChar ; length : Word);
var
  Index : Word;
begin
  for Index := 0 to length do
     output[Index] := input[Index];
end;
```

```
{Decodes input (unsigned char) into output (UINT4). Assumes len is
 a multiple of 4.}
procedure Decode(var output : Array of LongInt ; input : PChar ; len :
Word);
var
  i, j : Word;
  a,b,c,d : LongInt;

begin
  i := 0;
  j := 0;
  while j < len do
     begin
     a := LongInt(input[j]);
     b := LongInt(input[j+1]);
     c := LongInt(input[j+2]);
     d := LongInt(input[j+3]);
     output[i] := a or (b shl 8) or (c shl 16) or (d shl 24);
     Inc(i);
     Inc(j,4);
     end;
end;



procedure MD5_memset(output : PChar ; value : Integer ; length : Word);
var
  Index : Word;
begin
  for Index := 0 to length do
     output[Index] := Chr(value);
end;
procedure MD5Transform (var state : Array of LongInt ; block : PChar);
var
  a,b,c,d : LongInt;
  x : Array[0..15] of LongInt;
  sA,sB,sC,sD : string;
begin
  a := state[0];
  b := state[1];
  c := state[2];
  d := state[3];

  Decode (x, block, 64);

  FF (a, b, c, d, x[ 0], S11, $d76aa478);
  FF (d, a, b, c, x[ 1], S12, $e8c7b756);
  FF (c, d, a, b, x[ 2], S13, $242070db);
  FF (b, c, d, a, x[ 3], S14, $c1bdceee);
  FF (a, b, c, d, x[ 4], S11, $f57c0faf);
  FF (d, a, b, c, x[ 5], S12, $4787c62a);
  FF (c, d, a, b, x[ 6], S13, $a8304613);
  FF (b, c, d, a, x[ 7], S14, $fd469501);
  FF (a, b, c, d, x[ 8], S11, $698098d8);
  FF (d, a, b, c, x[ 9], S12, $8b44f7af);
  FF (c, d, a, b, x[10], S13, $ffff5bb1);
  FF (b, c, d, a, x[11], S14, $895cd7be);
  FF (a, b, c, d, x[12], S11, $6b901122);
  FF (d, a, b, c, x[13], S12, $fd987193);
  FF (c, d, a, b, x[14], S13, $a679438e);
  FF (b, c, d, a, x[15], S14, $49b40821);

  { Round 2 }
  GG (a, b, c, d, x[ 1], S21, $f61e2562);
  GG (d, a, b, c, x[ 6], S22, $c040b340);
  GG (c, d, a, b, x[11], S23, $265e5a51);
  GG (b, c, d, a, x[ 0], S24, $e9b6c7aa);
  GG (a, b, c, d, x[ 5], S21, $d62f105d);
  GG (d, a, b, c, x[10], S22,  $2441453);
  GG (c, d, a, b, x[15], S23, $d8a1e681);
  GG (b, c, d, a, x[ 4], S24, $e7d3fbc8);
  GG (a, b, c, d, x[ 9], S21, $21e1cde6);
  GG (d, a, b, c, x[14], S22, $c33707d6);
  GG (c, d, a, b, x[ 3], S23, $f4d50d87);
  GG (b, c, d, a, x[ 8], S24, $455a14ed);
  GG (a, b, c, d, x[13], S21, $a9e3e905);
  GG (d, a, b, c, x[ 2], S22, $fcefa3f8);
  GG (c, d, a, b, x[ 7], S23, $676f02d9);
  GG (b, c, d, a, x[12], S24, $8d2a4c8a);

  { Round 3 }
  HH (a, b, c, d, x[ 5], S31, $fffa3942);
  HH (d, a, b, c, x[ 8], S32, $8771f681);
  HH (c, d, a, b, x[11], S33, $6d9d6122);
  HH (b, c, d, a, x[14], S34, $fde5380c);
  HH (a, b, c, d, x[ 1], S31, $a4beea44);
  HH (d, a, b, c, x[ 4], S32, $4bdecfa9);
  HH (c, d, a, b, x[ 7], S33, $f6bb4b60);
  HH (b, c, d, a, x[10], S34, $bebfbc70);
  HH (a, b, c, d, x[13], S31, $289b7ec6);
  HH (d, a, b, c, x[ 0], S32, $eaa127fa);
  HH (c, d, a, b, x[ 3], S33, $d4ef3085);
  HH (b, c, d, a, x[ 6], S34,  $4881d05);
  HH (a, b, c, d, x[ 9], S31, $d9d4d039);
  HH (d, a, b, c, x[12], S32, $e6db99e5);
  HH (c, d, a, b, x[15], S33, $1fa27cf8);
  HH (b, c, d, a, x[ 2], S34, $c4ac5665);

  { Round 4 }
  II (a, b, c, d, x[ 0], S41, $f4292244);
  II (d, a, b, c, x[ 7], S42, $432aff97);
  II (c, d, a, b, x[14], S43, $ab9423a7);
  II (b, c, d, a, x[ 5], S44, $fc93a039);
  II (a, b, c, d, x[12], S41, $655b59c3);
  II (d, a, b, c, x[ 3], S42, $8f0ccc92);
  II (c, d, a, b, x[10], S43, $ffeff47d);
  II (b, c, d, a, x[ 1], S44, $85845dd1);
  II (a, b, c, d, x[ 8], S41, $6fa87e4f);
  II (d, a, b, c, x[15], S42, $fe2ce6e0);
  II (c, d, a, b, x[ 6], S43, $a3014314);
  II (b, c, d, a, x[13], S44, $4e0811a1);
  II (a, b, c, d, x[ 4], S41, $f7537e82);
  II (d, a, b, c, x[11], S42, $bd3af235);
  II (c, d, a, b, x[ 2], S43, $2ad7d2bb);
  II (b, c, d, a, x[ 9], S44, $eb86d391);

  state[0] := state[0] + a;
  state[1] := state[1] + b;
  state[2] := state[2] + c;
  state[3] := state[3] + d;

 {zeroize sensitive information.}
  MD5_memset (PChar(@x), 0, sizeof (x));
end;



procedure MD5Update(context : PMD5_CTX ; input : PChar ; inputLen :
Word);
var
  i : Word;
  index : Word;
  partLen : Word;
  w_temp : word;
  stemp : string;

begin
  {Compute number of bytes mod 64}
  index := Word( (Word(context^.count[0]) shr 3) and $3F);

  {Update number of bits}
  w_temp := (inputLen shl 3);
  context^.count[0] := context^.count[0] + w_temp;

  if context^.count[0] < w_temp then
     Inc(context^.count[1]);

  w_temp := (inputLen shr 29);
  context^.count[1] := context^.count[1] + w_temp;
  partLen := 64 - index;


  {Transform as many times as possible.}
  if inputLen >= partLen then
     begin
     MD5_memcpy(PChar(@context^.buffer[index]), PChar(input), partLen);
     MD5Transform (context^.state, context^.buffer);
     i := partLen;
     while (i + 63) < inputLen do
        begin
        MD5Transform (context^.state, @input[i]);
        i := i + 64;
        end;
     index := 0;
     end
  else
     i := 0;
  {Buffer remaining input}
  MD5_memcpy(@context^.buffer[index],@input[i],inputLen-i);
end;


procedure MD5Init(context : PMD5_CTX);
var
  PadIndex : Integer;
begin
  {Initialize the PADDING array}
  PADDING[0] := Chr($80);
  for PadIndex := 1 to 63 do
     PADDING[PadIndex] := Chr(0);

  {Initialize the context}
  context^.count[0] := 0;
  context^.count[1] := 0;
  context^.state[0] := $67452301;
  context^.state[1] := $efcdab89;
  context^.state[2] := $98badcfe;
  context^.state[3] := $10325476;
end;
procedure Encode(output : PChar ; input : Array of LongInt ; len :
Word);
var
  i, j : Word;
begin
  i := 0;
  j := 0;

  while j < len do
     begin
     output[j]   := Char(input[i] and $FF);
     output[j+1] := Char( (input[i] shr 8) and $FF) ;
     output[j+2] := Char( (input[i] shr 16) and $FF) ;
     output[j+3] := Char( (input[i] shr 24) and $FF) ;
     Inc(i);
     Inc(j,4);
     end;
  end;




 procedure MD5Final(digest : PChar; context : PMD5_CTX);
var
  bits : Array[0..8] of Char;
  Index : Word;
  padLen : Word;
  sTemp : string;
begin

  {Save number of bits}
  MD5_memset(bits,64,8);
  Encode (bits, context^.count, 8);

  {Pad out to 56 mod 64.}
  index := Word((context^.count[0] shr 3) and $3f);
  Str(index,sTemp);

  if index < 56 then
     padLen := (56 - index)
  else
     padLen := (120 - index);

  Str(padLen,sTemp);

  MD5Update(context, PADDING, padLen);

  { Append length (before padding) }
  MD5Update (context, bits, 8);
  {Store state in digest }
  Encode (digest, context^.state, 16);

  { Zeroize sensitive information. }
  MD5_memset(PChar(context), 0, sizeof(context));
end;


function tohex(a: integer):string;

var MSnibble,LSnibble : integer; temp : string;
begin
  MSnibble := 0;
  LSnibble := 0;
  temp := '';
  LSnibble := a AND 15;
  MSnibble := a AND 240;
  MSNibble := MSNibble shr 4;
  temp := copy(hexlisting,msnibble,1);
  temp := temp + copy(hexlisting,lsnibble,1);
  tohex := temp;
end;  { tohex }



function  MD5ProduceDigest(PassText : string) : string;
var

  context : MD5_CTX;
  sDigest : string;
  sTemp : string;
  szMessage : Array[0..255] of Char;
  digest : Array[0..16] of Char;
  len : Word;
  i,x : integer;

begin
  szMessage[0] := Chr(0);
  StrPCopy(szMessage,PassText);
  len := StrLen(szMessage);

  MD5Init(@context);
  MD5Update(@context,szMessage,len);
  MD5Final(digest, @context);
  sTemp := '';
  sdigest :='';
    for i := 0 to 15 do
     begin
        x := Integer(digest[i]);
        stemp := tohex(x);

        sDigest := sDigest + sTemp;
     end;
  md5producedigest := sDigest;
end;
```

9.13. Test Run of Smart Card Simulation

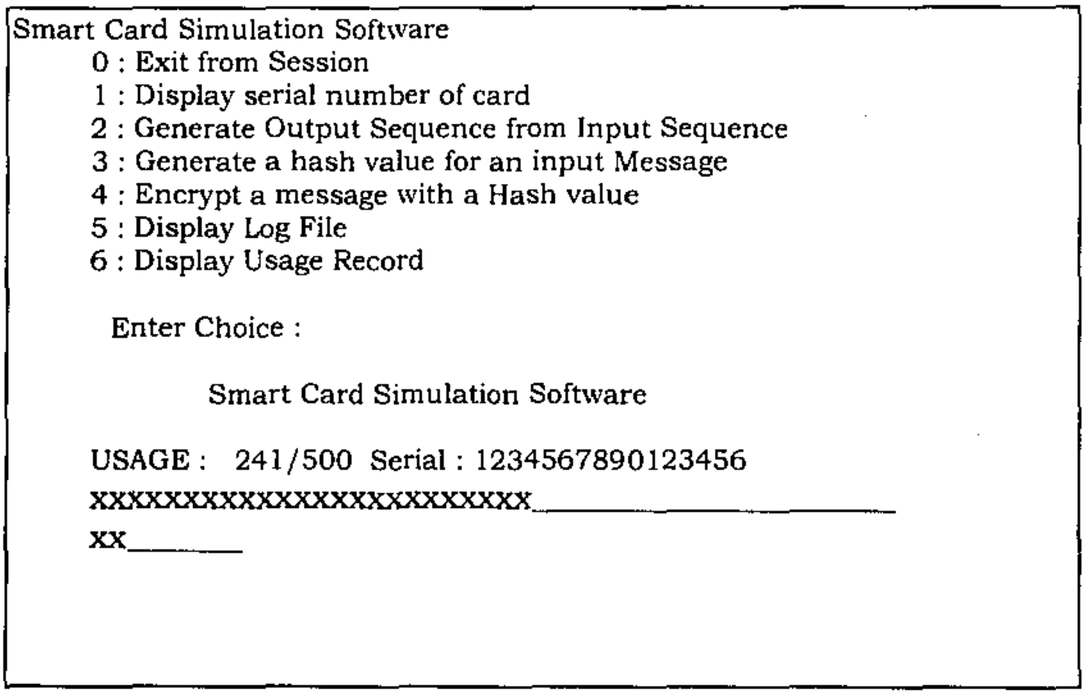
9.13.1. Initial Screen to Request Password



```
Smart Card Simulation Software


Input the password now : 123
```

If the password is entered incorrectly more than three times the card self destructs by destroying the randomiser, selection set and startpoint files. This option is implemented but disabled in the simullation.

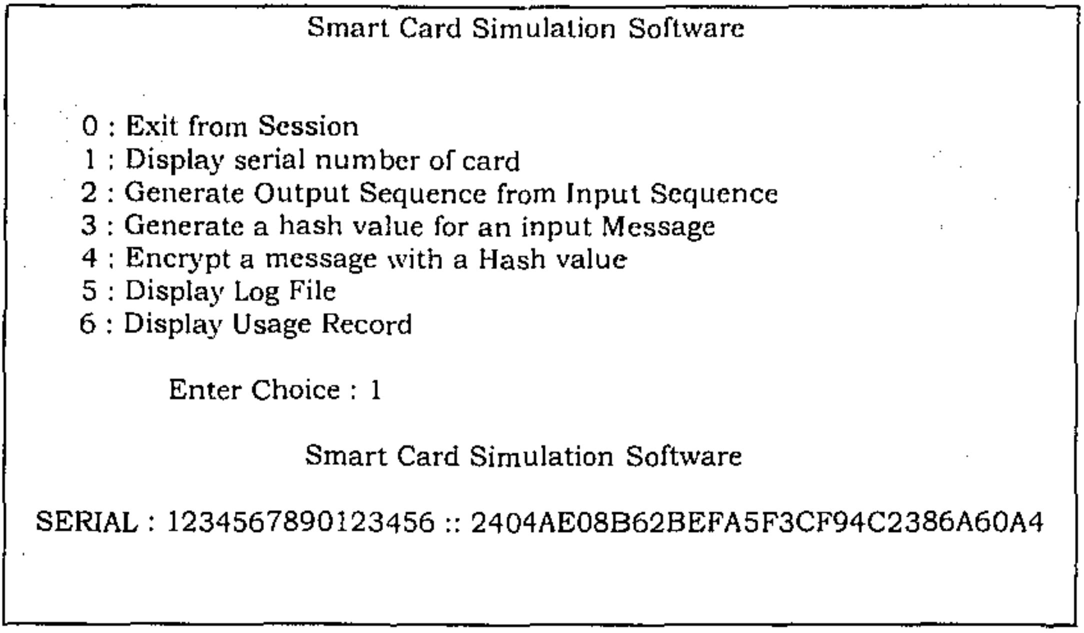
9.13.2. Menu Screen



```
Smart Card Simulation Software
       0 : Exit from Session
       1 : Display serial number of card
       2 : Generate Output Sequence from Input Sequence
       3 : Generate a hash value for an input Message
       4 : Encrypt a message with a Hash value
       5 : Display Log File
       6 : Display Usage Record

         Enter Choice :

                  Smart Card Simulation Software

       USAGE :  241/500  Serial : 1234567890123456
       xxxxxxxxxxxxxxxxxxxxxxxx_____________________
       xx______
```

Displays permitted options and the current usage state. If no choice is made within a set time the session enters the 'time-out'state and is closed.

9.13.3. Display Card Serial Number

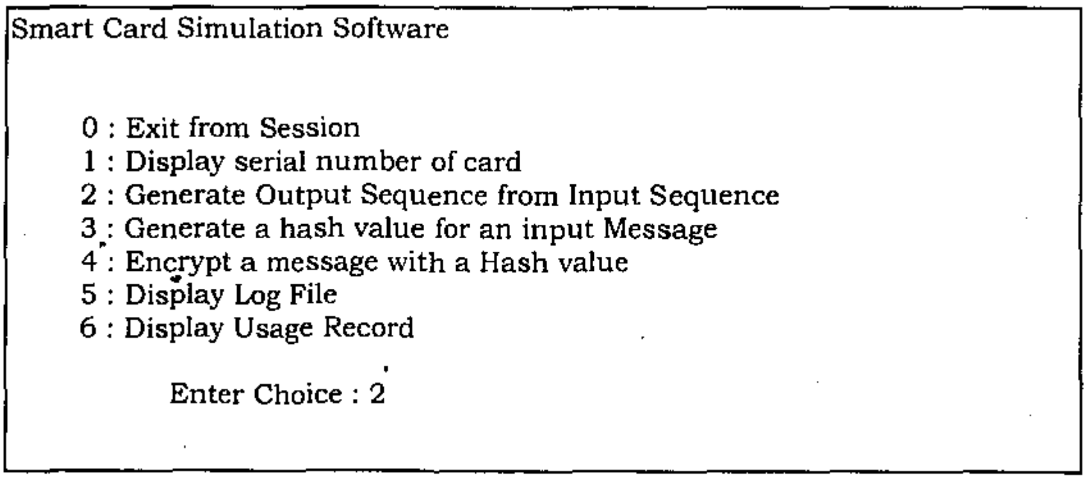


```
                    Smart Card Simulation Software


  0 : Exit from Session
  1 : Display serial number of card
  2 : Generate Output Sequence from Input Sequence
  3 : Generate a hash value for an input Message
  4 : Encrypt a message with a Hash value
  5 : Display Log File
  6 : Display Usage Record

        Enter Choice : 1

                    Smart Card Simulation Software

SERIAL : 1234567890123456 :: 2404AE08B62BEFA5F3CF94C2386A60A4
```

The number '1234567890123456' is the serial number of the card devised by the manufacturer.

The number '2404AE08B62BEFA5F3CF94C2386A60A4' is the serial number derived from the randomiser written into the ROM on the Smart Card.

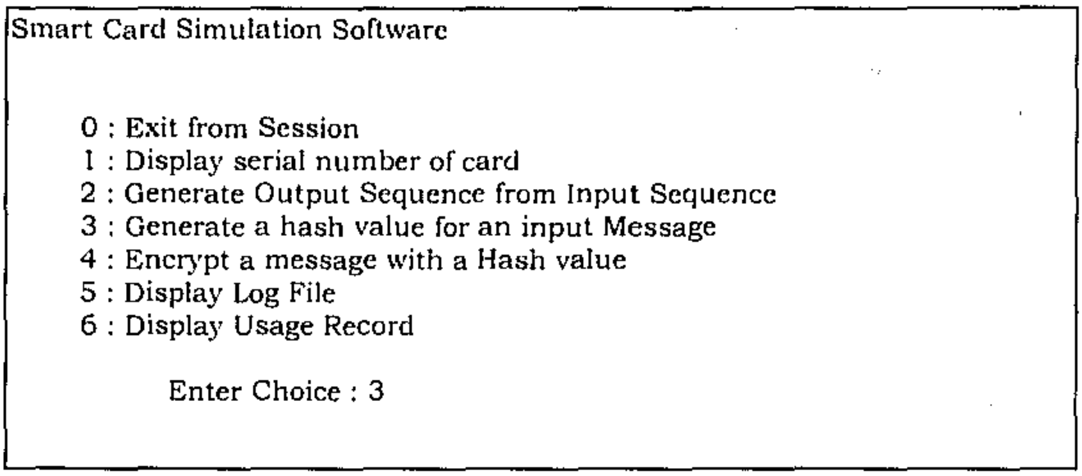
9.13.4. Generate An OutPut Sequence From And Input Sequence



```
Smart Card Simulation Software


    0 : Exit from Session
    1 : Display serial number of card
    2 : Generate Output Sequence from Input Sequence
    3 : Generate a hash value for an input Message
    4 : Encrypt a message with a Hash value
    5 : Display Log File
    6 : Display Usage Record

          Enter Choice : 2
```

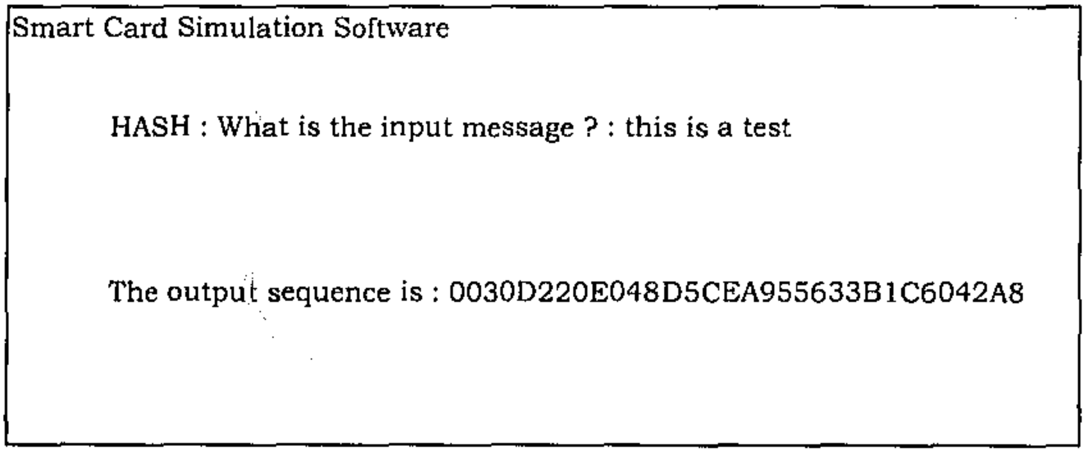
The output sequence is currently fixed. It demonstrates the processing of an input sequence into an output sequence.

9.13.5. Hashing an Input Sequence

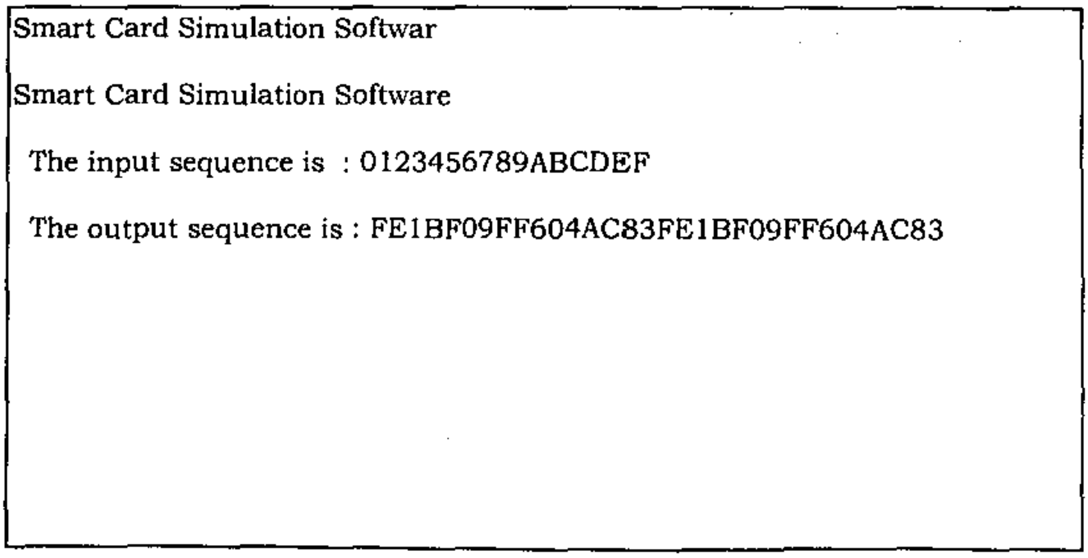


```
Smart Card Simulation Software


     0 : Exit from Session
     1 : Display serial number of card
     2 : Generate Output Sequence from Input Sequence
     3 : Generate a hash value for an input Message
     4 : Encrypt a message with a Hash value
     5 : Display Log File
     6 : Display Usage Record

            Enter Choice : 3
```

The input sequence provided by the user is 'this is a test' which is hashed to produce an output sequence of '0030D220E048D5CEA955633B1C6042A8'.

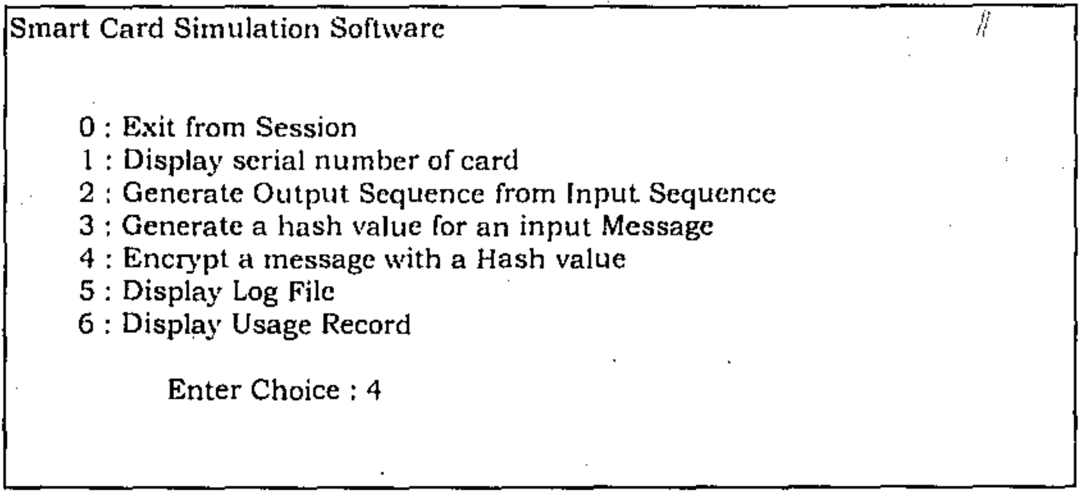


```
Smart Card Simulation Software


        HASH : What is the input message ? : this is a test




        The output sequence is : 0030D220E048D5CEA955633B1C6042A8
```



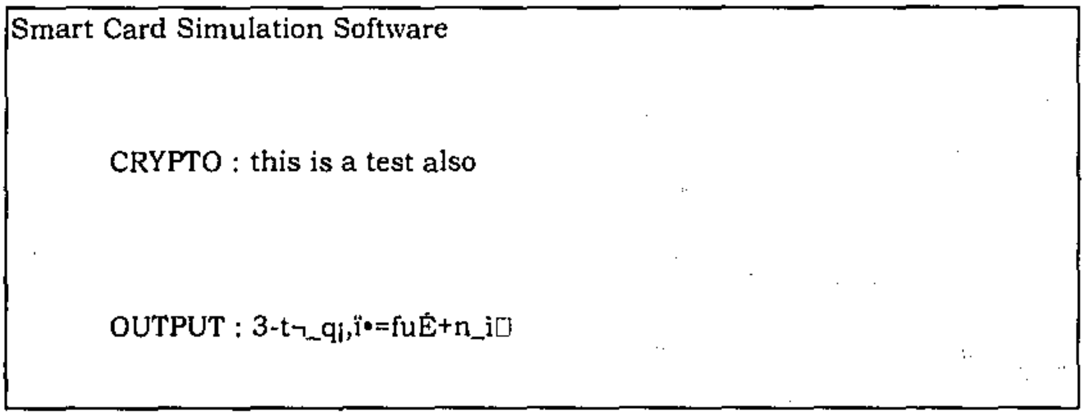
```
Smart Card Simulation Softwar

Smart Card Simulation Software

 The input sequence is  : 0123456789ABCDEF

 The output sequence is : FE1BF09FF604AC83FE1BF09FF604AC83
```

9.13.6. Encrypting An Input Sequence



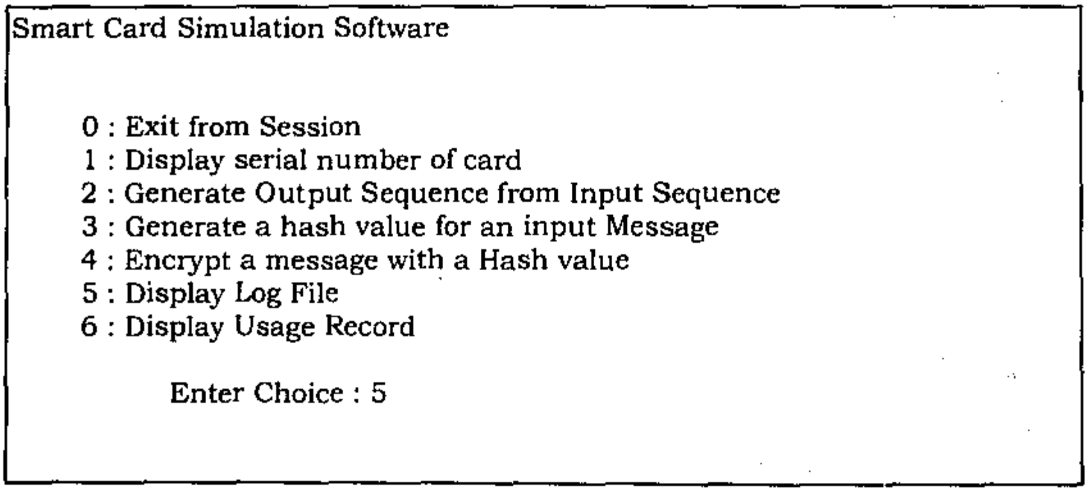
```
Smart Card Simulation Software


      0 : Exit from Session
      1 : Display serial number of card
      2 : Generate Output Sequence from Input Sequence
      3 : Generate a hash value for an input Message
      4 : Encrypt a message with a Hash value
      5 : Display Log File
      6 : Display Usage Record

            Enter Choice : 4
```

The input sequence is 'this is a test also' which is hashed to provide an input sequence to the key generator. This sequence is used to select the start point in the randomiser and a key of the requisite length is determined. The key sequence and the input sequence are processed bitwise with an 'exclusive-or process' and the output sequence is determined.

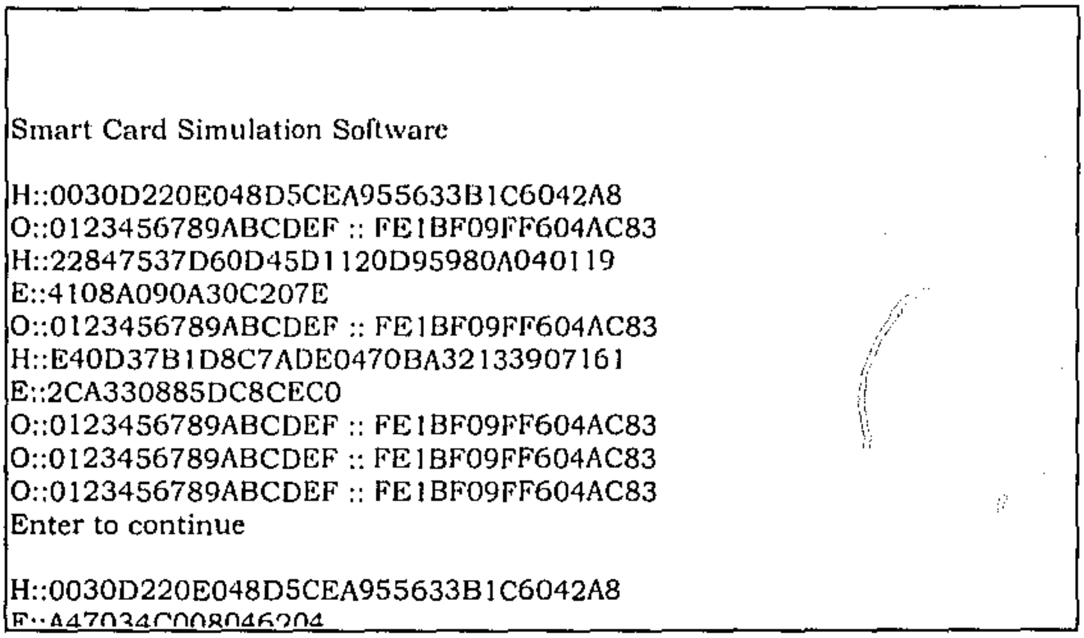


```
Smart Card Simulation Software



        CRYPTO : this is a test also




        OUTPUT : 3-t¬_q¡,ï•=fuÉ+n_i□
```

9.13.7. Display The Transaction Log



```
Smart Card Simulation Software


      0 : Exit from Session
      1 : Display serial number of card
      2 : Generate Output Sequence from Input Sequence
      3 : Generate a hash value for an input Message
      4 : Encrypt a message with a Hash value
      5 : Display Log File
      6 : Display Usage Record

            Enter Choice : 5
```



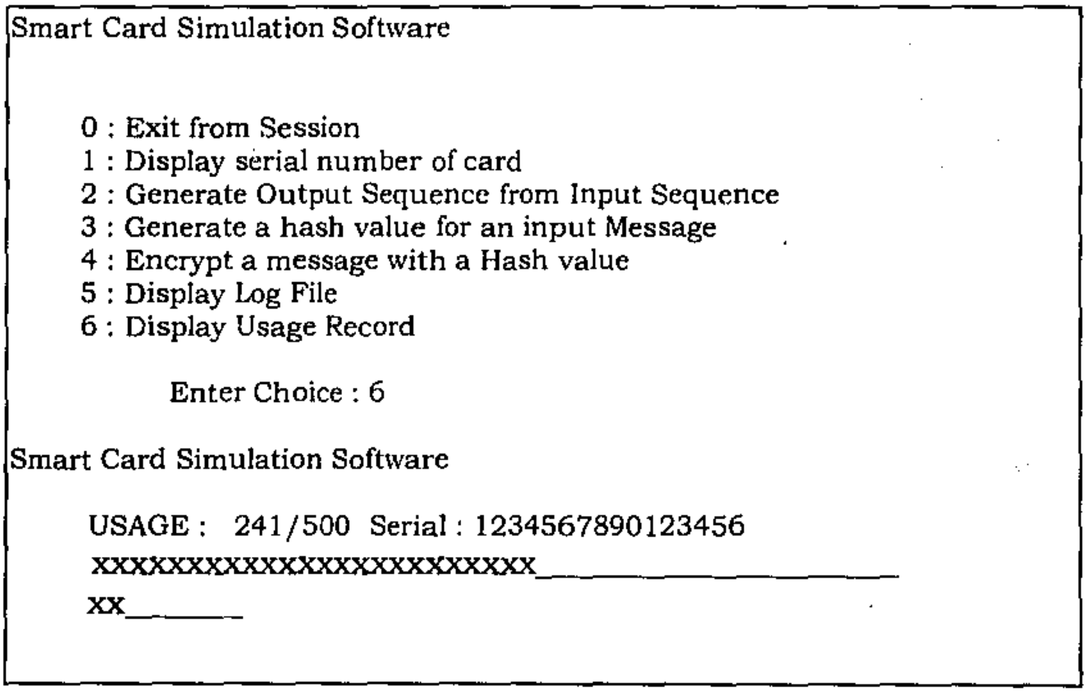
```
Smart Card Simulation Software

H::0030D220E048D5CEA955633B1C6042A8
O::0123456789ABCDEF :: FE1BF09FF604AC83
H::22847537D60D45D1120D95980A040119
E::4108A090A30C207E
O::0123456789ABCDEF :: FE1BF09FF604AC83
H::E40D37B1D8C7ADE0470BA32133907161
E::2CA330885DC8CEC0
O::0123456789ABCDEF :: FE1BF09FF604AC83
O::0123456789ABCDEF :: FE1BF09FF604AC83
O::0123456789ABCDEF :: FE1BF09FF604AC83
Enter to continue

H::0030D220E048D5CEA955633B1C6042A8
E::A47034C008046204
```

The output sequence is 'O::0123456789ABCDEF :: FE1BF09FF604AC83' where the initial character 'O::' indicates an output.

The hash sequence is 'H::0030D220E048D5CEA955633B1C6042A8' where the initial character 'H::' indicates an hashed output. The input sequence is not stored.

The encryption sequence is 'E::A47034C008046204' where the initial character 'E::' indicates an hashed output. The input sequence is not stored.and the output is stored in hexadecimal format.

9.13.8. Display Current Usage Record



```
Smart Card Simulation Software


     0 : Exit from Session
     1 : Display serial number of card
     2 : Generate Output Sequence from Input Sequence
     3 : Generate a hash value for an input Message
     4 : Encrypt a message with a Hash value
     5 : Display Log File
     6 : Display Usage Record

           Enter Choice : 6

Smart Card Simulation Software

     USAGE :  241/500  Serial : 1234567890123456
     xxxxxxxxxxxxxxxxxxxxxxxx___________________
     xx_____
```

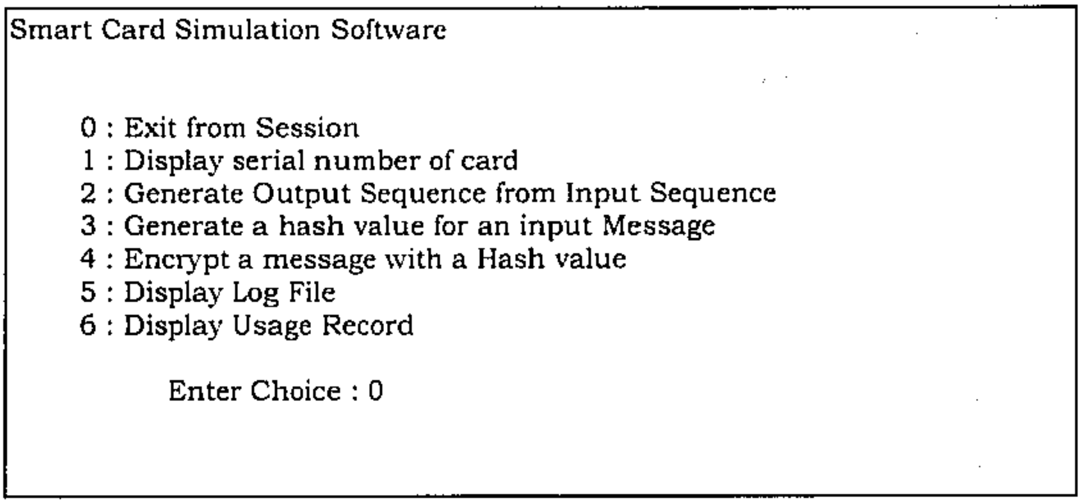
The usage record consists of a session count (241) and maximum permitted uses (500). The card serial number is displayed and a bar graph consisting of a top row which count the tens and a shorter bottom row which counts from 0 to 9. For example:

```
xxxxxxxxxxxxxxxxxxxxxxxx__________________________
xx________
```

indicates 241.

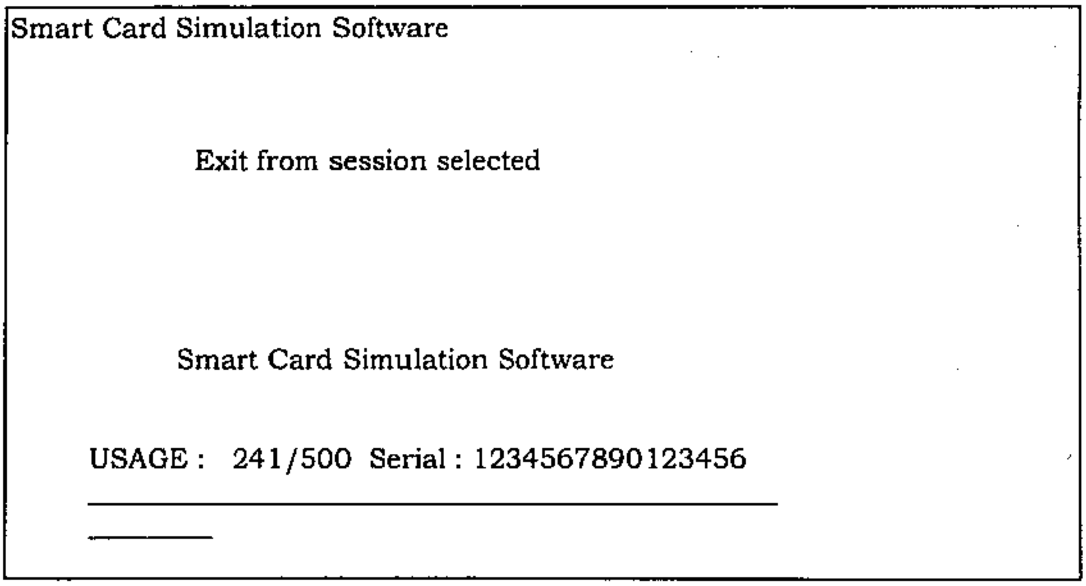
The card will cease functioning when the counter exceeds 500.

9.13.9. Exit From The Session Selected



```
Smart Card Simulation Software


    0 : Exit from Session
    1 : Display serial number of card
    2 : Generate Output Sequence from Input Sequence
    3 : Generate a hash value for an input Message
    4 : Encrypt a message with a Hash value
    5 : Display Log File
    6 : Display Usage Record

          Enter Choice : 0
```

The exit from the session displays the session count and disables the input. A complete new session must be commenced.



```
Smart Card Simulation Software


            Exit from session selected




          Smart Card Simulation Software


    USAGE :  241/500  Serial : 1234567890123456
    __________________________________________
    ________
```

9.14. Data Visualisation

```
    program linearwalk;
    generate an array of longintegers and store in a file.
     each array of longint is generated by starting a random generator
     at the centre of the screen and counting the number of hits in each
     pixel until a border value is detected then it stops.
     the max longint value is then scaled to permit contouring of the
     display.
    }

uses
  printer,       { hardcopy of screen display }
  crt,           { text screen functions }
  graph;         { graphics screen functions }

const
  maxarray    = 401;      { array length }
  arraycentre = 201;      { midpoint of array length }
  ESC         = #$1B;
  portrait    = 0;        { printer output orientation }
  landscape   = 1;

type
  lintarray = array[1..maxarray] of longint;
  lintfile  = file of lintarray;
  lfilename = string[80];

var
  store     : lintfile;    { file of longinteger arrays }
  storename : lfilename;

  anyarray,            { input array }
  gross                { total array }
        : lintarray;
  ch : char;           { loop response }

  increment,
  max,
  size : longint;

  row,
  count,
  num,
  current : integer;   { arrayindex for insertion }

  grflag,              { graphics ok flag }
  finish               { terminate loop flag }
        : boolean;
{=========================================================}

procedure printgraph(Mode, direction : integer);
var i,j,k,m,
    msb,lsb,Maxx,Maxy : integer;

begin
```

```
  maxx := getmaxx;
  maxy := getmaxy;
  setviewport(0,0,maxx,maxy,false);
  write(lst, ESC,'A',#$07); { on printer set line spacing to 7/72 inch }

  case direction of
    portrait :  begin
                  lsb := maxx AND $00ff;  { maxx mod 256 }
                  msb := maxx shr 8;     { int maxx / 256 }
                  for j := 0 to maxy div 8 do
                  begin
                    write(lst,ESC, char(mode),char(lsb),char(msb));
                    for i := 0 to maxx do
                    begin
                      m := 0;
                      for k := 0 to 7 do
                      begin
                        m := m shl 1; { shift m left 1 bit }
                        if getpixel(i, j*8 + k) <> 0 then inc(m);
                      end;  { for k }
                      write(lst, char(m));

                    end; { for i }
                    write(lst, #$0D, #$0A);
                  end; { for j }

                end; { portrait }
    landscape : begin
                  lsb := maxy AND $00ff;  { maxy mod 256 }
                  msb := maxy shr 8;     { int maxx / 256 }

                  j := 0;
                  repeat
                    write(lst,ESC,'*', char(mode),char(lsb),char(msb));

                    for i := maxy downto 0 do
                    begin
                      m := 0;
                      for k := 0 to 7 do
                      begin
                        m := m shl 1; { shift m left 1 bit }
                        if getpixel(j+ k,i) <> 0 then inc(m);
                      end;  { for k }
                      write(lst, char(m));

                    end; { for i }
                    write(lst, #$0D, #$0A);
                    inc(j,8);

                  until j >= maxx-1;
                end; { landscape }
  end; { case }
  write(lst,#$0C);  { form feed }
end;  { printgraph }

procedure cleararray(var a : lintarray);
var count : integer;
begin
  for count := 1 to maxarray do
    a[count] := 0;
end;
```

```
procedure writearray(var a : lintarray);
var count : integer;
begin
  for count := 1 to maxarray do
  begin
    if count = 200 then textcolor(yellow)
    else textcolor(white);
    write(a[count],' ');
  end;
end;  { writearray }

procedure beep;
begin
  sound(500);
  delay(500);
  nosound;
end;

procedure readfile(var s : lintfile);
var count : integer;
    a : lintarray;

begin
   reset(s);
   while not eof(s) do
   begin
     read(s,a);
     writearray(a);
     readln;
   end;
end; { readfile }

procedure dorun(var s : lintfile;
                var a : lintarray );
begin
  size := FileSize(s);
  Seek(s,size);

  finish := false;
  cleararray(a);
  randomize;
  current := arraycentre;
  While not finish do
  begin
     num := random(2);
     case num of
        1 : begin
              inc(current);
              inc(a[current]);
            end;
        0 : begin
              dec(current);
              inc(a[current]);
            end;
     end; { case }

    if (current = 1) or (current = maxarray) then finish := true;

    if keypressed then break;
  end; { while not finish }
  write(s,a);
end;    { dorun }

procedure getmax(var s : lintfile;
                 var m : longint);
var temp : lintarray;
    max : longint;
    count : integer;
begin
  reset(s);
  max := 0;
  while not eof(s)do
  begin
    read(s,temp);
    for count := 1 to maxarray do
    begin
      if temp[count] >max then max := temp[count];
    end; { for }

  end;
  m:= max;
end; { getmax }

procedure initialisegraphics(var gflag : boolean);
var
  grDriver: Integer;
  grMode: Integer;
  ErrCode: Integer;
begin
  grDriver := Detect;
  InitGraph(grDriver, grMode,'c:\language\bp\bgi ');
 { setgraphmode(grmode); 0 - vga low   640*200 2 pages
                         1 - vga med   640*350 2 pages
                         2 - vga high  640*480 1 page}

  ErrCode := GraphResult;
  if ErrCode = grOk then
     gflag := true
  else
  begin
    Writeln('Graphics error:', GraphErrorMsg(ErrCode));
    gflag := false;
  end;    { end if errcode }
end; { initialisegraphics }

procedure putgraticule;
begin
 rectangle(0,0,getmaxx,getmaxy);
 floodfill(10,10,white);
 setcolor(black);
 Line(99,10,99,459);     { vertical left line }
 Line(502,10,502,459);   { vertical right line }

 line(100,459,501,459); { base line }
 line(301,10,301,459);   { central line }


 outtextxy(100,465,'-200');  { base line indices }
 outtextxy(297,465,'0');
 outtextxy(499,465,'200');
```

```
end; { put graticule }

procedure display(var a: lintarray;
                     x,y,r : integer);
var count, scale,
    baselinex,
    baseliney : integer;
begin
  setcolor(black);
  baselinex := x;
  baseliney := r * y;
  scale  := 20;
  for count := 1 to maxarray do
  begin
    putpixel(baselinex + count, baseliney- (a[count] div scale), black);


  end; { for }

end;  { display }

procedure displayline(a : lintarray);
var count, scale,
    baselinex,
    baseliney : integer;
begin
  setcolor(black);
  baselinex := 100;
  baseliney := getmaxy - 50;

  scale  := 10;

  for count := 1 to maxarray do
  begin
    putpixel(baselinex + count, baseliney - (a[count] div scale),
black);


  end; { for }

end;  { displayline }

procedure keeptotals(var a,g : lintarray);
var count : integer;
begin
  for count := 1 to maxarray do
    g[count] := a[count] + g[count];

end;



{========================================================}
begin

  clrscr;

  storename := 'd:\programs\datavis\store.lni';
  assign(store,storename);
  reset(store);
  initialisegraphics(grflag);
  if grflag then            { graphics is ok }
  begin
    rectangle(1,1,getmaxx,getmaxy);
    setviewport(3,3,getmaxx-3,getmaxy-3,false);
    clearviewport;
    floodfill(10,10,white);
    while not finish do
    begin
      if keypressed then ch := readkey;  { test for quit }
      if (ch = 'Q') or (ch = 'q') then finish := true;

      reset(store);    { Prepare input file }
      row := 0;
      putgraticule;
      cleararray(gross);

      while not eof(store) do
      begin
        inc(row);
        cleararray(anyarray);
        read(store,anyarray);
        keeptotals(anyarray,gross);{ keep totals }
                        { x      y           color }
        display(anyarray,100,45 , row);

      end;  { while not eof }
      if eof(store) then finish := true;
    end; { while }
    readln;
  end;  { if grflag }

  ch := readkey;

    if (ch = 'P') or (ch = 'p') then printgraph(4,landscape);
    ch := readkey;

  clearviewport;
  putgraticule;
  displayline(gross);

  ch := readkey;

  if (ch = 'P') or (ch = 'p') then printgraph(4,landscape);
    ch := readkey;
    read;   { pause until key pressed }

  closegraph;
  close(store);
end.  { main }
```

```
program dwc;

{$I d:\programs\datavis\DWCLIB.PAS}  { include library file }


{-------------------------- main ---------------------------------------
-}
begin { dwc }
  cleartextscreen;
  menu;
  getfiles(session,start);

  if start then
  begin
    loop1 := 0;
    initialisegraphics(grflag);   { test for graphics adapter }
    if grflag then    { graphics available }
    begin
      finish := false;
      While not finish do
      begin
        dograticule(session);                  { set up screen }
        initscreenrec(session);          { set up default values }
        doinput(session);          { alter values as required }

        if session.shape = 1 then              { size the target }
          squarecountpixels(session)
        else
          circlecountpixels(session);

        initview(session.screen.g);
        randomize;                       { initialise the randomiser }
        for loop1 := 1 to session.maxrun do
        begin
          session.loop    := loop1;
          current := session.centre;      { reset to centre }
          restart := false;
          loop2   := 0;

          case session.shape of    { 1 = square 2 = circle }
            1 : case session.numberbits of
                  2 : dosquare2(session);
                  3 : dosquare3(session);
                  4 : dosquare4(session);
                end;
            2 : case session.numberbits of
                  2 : dotarget2(session);
                  3 : dotarget3(session);
                  4 : dotarget4(session);
                end; { case }
          end; { case }
          case session.numberbits of
            2 : dorun2(session,loop2);
            3 : dorun3(session,loop2);
            4 : dorun4(session,loop2);
          end;  { case }

          density(session);
          dotext(session,loop1);

        end; { for loop }
        beep;

        while not keypressed do     { quit  or print result }
        begin
          ch := readkey;
          case ch of
          'R','r' : begin  { restart }
                      finish := false;
                      break; { leave while state }
                    end;
          'Q','q' : begin  { quit }
                      finish := true;
                      break; { leave while state }
                    end;
          'P','p' : printgraph(4);

          else finish := true;
          end;  { case }
        end; { while not keypressed }

      end; { while not finish }
      closegraph;   { When program over close graphics }
    end;  { if grflag }
  end; { if start }
  closefiles(session);
  endmessage;
end.  { dwc }
```

```
{ Drunkards Walk Library functions }
Uses
    dos,                       { use dos functions for file and time }
    printer,                   { output to printer }
    graph,                     { use graphics window functions }
    crt;                       { use text window functions }

const
    shortdelay = 1500;

    blank6str  = '000000';
    blank40str = '0000000000000000000000000000000000000000';

    headerstring = 'Data Visualisation Program';

    inputname    = 'intinput.dat';   { default values }
    outputname   = 'intoutpt.dat';

    days : array [0..6] of String[9] =   { day conversion }
           ('Sunday','Monday','Tuesday',
            'Wednesday','Thursday','Friday',
            'Saturday');

    months : array[1..12] of string[3] =  { month conversion }
             ('JAN','FEB','MAR','APR','MAY','JUN',
              'JUL','AUG','SEP','OCT','NOV','DEC');

     maxruns   =   1;    { default maximum number of runs permitted }

     minradius =  50;    { default smallest size of target }
     midradius = 100;    { default size of target }
     maxradius = 200;    { default largest  size of target }


     tcol1 =  10;    { text x position values }
     tcol2 = 150;
     tcol3 = 250;
     tcol4 = 400;

     trow  =  10;    { text y position values }

     textmodetext = yellow;         { text mode text colour }
     textmodeback = blue;           { text mode background }

     { graphics mode colours }
     linecolour   = magenta;

     labelcolour  = lightgray;      { text window labels }
     textcolour   = lightcyan;      { text window results}

     blankcolour  = green;          { blanking colour }
     changecolour = white;          { change text colour }
     errorcolour  = red;            { colour for errors }

     gratcolour   = yellow;         { target colour }
     areacolour   = green;          { overwrite colour }
     tracecolour  = lightblue;      { trace colour }
     crosscolour  = red;

   gbackcolour  = lightgray;        { backgound colour }
```

```
    tbackcolour  = blue;            { text window background }
    ibackcolour  = blue;
    hbackcolour  = yellow;

    degree     = 0.017453292;    { degree in radians }

    ESC        = #$1B;           { hexadecimal equivalent of escape key }

    onesize    = 255;            { maxsize of 1d array }

{------------------ data store types ------------------------------ }

type
     values = (total,min,avg,max,counter);   { array index }

     valuesarray = array[values] of longint; { data store }
     realarray   = array[values] of real;


     onearray    = array[0..onesize] of longint; { 1d array }
     twoarray    = array[1..2] of shortint;      { 2d array }
     threearray  = array[1..3] of shortint;      { 3d array }
     fourarray   = array[1..4] of shortint;      { 4d array }

{------------- file handling types ------------------------------}
     direction = (input,output,inout,none);   { file data direction }

     filename    = string[40];
     sintfile    = file of shortint;          { data file }

     choicerecord = record  { progam choices storage }
                      inputfile,
                      outputfile : sintfile;

                      inputfilename,
                      outputfilename : filename;
                      flow : direction
                    end;

 {------------- graphics types -----------------------------------}
     numstr = string[20];     { temporary strings }
     nstr   = string[5];
     view   = record
                port : viewporttype;
                fill : integer;
              end;  { view }
     screens = record
                 g,   { graphics }
                 i,   { input }
                 t,   { text }
                 h    { help }
                   : view ;
               end;

     screenrecord = record   { screen data storage }
                      screen      : screens;
                      centre      : pointtype;
                      tradius     : longint; { size from centre }

                      step,         { number of pixels increment }
                      maxrun,       { maximum iterations }
```

```
                       shape,        { 1 = square 2 = circle }
                       numberbits : integer;

                       loop,        { iteration count }
                       targetarea  : longint; { area in circle }

                       aveexpblue,
                       greenarea   : real;

                       totalbits,      { total bits tested in all runs }

                       expectedblue,   { calculated value by kaye }
                       bluerunarea,    { area of run }
                       bluerunnumber'  { number of digits }
                                         : longint;

                       bluerundensity    : real;


                       bluecount,
                       bluearea          : valuesarray;
                       bluedensity       : realarray;    {  storage
arrays }
                       bluedirection     : valuesarray;

                       usefile,                      { file input/output
}
                       overlap           : boolean; { overlap data }

                       filechoice        : choicerecord;

                       timestr,
                       datestr           : numstr;
                     end; { end screen record }


     vector = record     { direction of trace }
                s,f : pointtype;
              end;
var  {-----------------global variables -----------------------}
     start,
     grflag,
     finish,
     restart : boolean;

     current,
     next    : pointtype;       { determine direction }

     Run,
     loop1,
     loop2   : integer;


     count : longint;

     pixelcolor : word;


     session  : screenrecord;    { common data for graphics }

     ch      : char;            { user response }
```

```
{-------------------- Functions -----------------------------------}
function LeadingZero(w : Word) : String;
var         { place a leading zero and convert time to string }
  s : String;
begin
  Str(w:0,s);
  if Length(s) = 1 then
    s := '0' + s;
  LeadingZero := s;
end;


function num2str(a : longint):numstr;
var
  temp1,
  temp2 : longint;
  c     : numstr;
begin     { convert integer to string for display }
  c := ''; temp1 := a; temp2  := 0;
  if a = 0 then c :='0';

  temp1 := abs(temp1);

  while temp1 <> 0 do
  begin
    temp2 := temp1 mod 10;
    c := chr(temp2 + 48) + c;
    temp1 := temp1 div 10;
  end; { while }

  if a < 0 then c := '-'+ c;
  num2str := c;
end;   { num2str }

procedure beep;
begin
    sound(400);
    delay(100);
    nosound;
end;  { beep }

function getnum(a : nstr; p,q : integer): integer;
var i,code : integer;
    tch : char;
begin
  while (ord(tch) <> 13) do
  begin
    tch := readkey;
    case ord(tch) of
       48..57 : begin
                  a := a + tch;
                  outtextxy(q,trow * p,a);
                end;
           13 : break;
    end; { case }
  end; { While }
  val(a,i,code);
  getnum := i;
end; {getnum}
```

```
function getstr(p,q : integer): filename;
var s : filename;
    code : integer;
    tch : char;
begin
  s := '';
  while (ord(tch) <> 13) do
  begin
    tch := readkey;
    case ord(tch) of
       32..122 : begin
                  s := s + tch;
                  outtextxy(p,trow * q,s);
                end;
           13 : break;
    end; { case }
  end; { While }
  getstr := s;
end;  { getnum }



{--------------------- Procedures ------------------------------------}
procedure initview(v : view);
begin
  with v do
  begin
    setviewport(port.x1,port.y1,port.x2,port.y2,port.clip);
    clearviewport;
  end; { with v }
    setcolor(v.fill);
    floodfill(v.port.x1,v.port.y1,linecolour);
end; { initview }

procedure circlecountpixels(var g : screenrecord);
var x,
    y     : integer; { count pixels in target }
    t     : longint;
    c     : word;
    count : boolean;
    range : integer;
begin
  with g do
  begin
    range := g.tradius + 1;
    initview(g.screen.g);

    setcolor(gratcolour);

    setfillstyle(solidfill,gratcolour); { draw and fill target }
    fillellipse(centre.x,centre.y, tradius,tradius);

    t := 0;    { initialise count }
    for x := centre.x - range to centre.x + range do
      for y := centre.y - range to centre.y + range do
      begin
        c := getpixel(x,y);         { test pixel colour }

        if c = gratcolour  then     { if yellow then part of target }
    begin
     inc(t);
```

```
          putpixel(x,y,areacolour);      { areacolour when counted }
        end;
        if c = gbackcolour then count := false;

    end;
    g.targetarea := t;
  end;  { with g }
end; { countpixels }


procedure squarecountpixels(var g : screenrecord);
var x,
    y     : integer; { count pixels in target }
    t     : longint;
    c     : word;
    count : boolean;
    range : integer;
    rect  : array[1..4] of pointtype;
begin

  range := g.tradius + 1;
  initview(g.screen.g);
  with g do
  begin
    setcolor(gratcolour);

    setfillstyle(solidfill,gratcolour); { draw and fill target }

    rect[1].x := (centre.x - tradius);
    rect[1].y := (centre.y - tradius);
    rect[2].x := (centre.x + tradius);
    rect[2].y := (centre.y - tradius);
    rect[3].x := (centre.x + tradius);
    rect[3].y := (centre.y + tradius);
    rect[4].x := (centre.x - tradius);
    rect[4].y := (centre.y + tradius);
    { print target }
    FillPoly(SizeOf(rect) div SizeOf(PointType), rect);

    t := 0;     { initialise count }
    for x := centre.x - range to centre.x + range do
      for y := centre.y - range to centre.y + range do
      begin
        c := getpixel(x,y);         { test pixel colour }

        if c = gratcolour  then     { if yellow then part of target }
        begin
          inc(t);
          putpixel(x,y,areacolour);      { areacolour when counted }
        end;
        if c = gbackcolour then count := false;

    end;
    g.targetarea := t;
  end;  { with g }

end; { countpixels }


procedure cleartextscreen;
begin
```

```
  textcolor(textmodetext);
  textbackground(textmodeback);
  clrscr;
  gotoxy(20,1);
  write(headerstring);
end;  { cleartextscreen }

procedure endmessage;
begin
  cleartextscreen;
  gotoxy(15,7);
  write('This Program has terminated normally.');
  delay(shortdelay);
end; { endmessage }

procedure closefiles(g : screenrecord);
begin
  if g.usefile then
  begin
    cleartextscreen;
    if (g.filechoice.flow = input) or
       (g.filechoice.flow = inout) then
    begin
      close(g.filechoice.inputfile);
      gotoxy(10,5);
      write(g.filechoice.inputfilename, ' closed');
    end;
    if (g.filechoice.flow = output) or
       (g.filechoice.flow = inout) then
    begin
      close(g.filechoice.outputfile);
      gotoxy(10,7);
      write(g.filechoice.outputfilename, ' closed');
    end;
    delay(shortdelay);
  end; { if }
  writeln;
end; { closefiles }

procedure menu;
begin
  gotoxy(15,3); Write('Choice 1 : NONE <Default>');
  gotoxy(15,4); write('       2 : Input');
  gotoxy(15,5); write('       3 : Output');
  gotoxy(15,6); write('       4 : Input and Output');
  gotoxy(15,7); write('       C : Continue');
  gotoxy(15,8); Write('       Q : EXIT');
  writeln;
end; { menu }

procedure filetest(var f  : sintfile;
                   var fn : filename;
                       c  : integer;
                   var cf : boolean);

begin

  if length(fn) > 0 then
  begin
9.15.  assign(f,fn);
```

```
  case c of
    1 : reset(f);
    2 : rewrite(f);
  end; { case }
{$I+}

  case ioresult of
    0 : writeln(fn,' : is available');
    2 : writeln(fn,' : File NOT found');
    3 : writeln(fn,' : Path NOT found');
    5 : writeln(fn,' : File Access Denied');
   12 : writeln(fn,' : Invalid File Access Code');
  end; { case }
  if ioresult <> 0 then cf := false;
 end; { if }
end;  { testfile }


procedure getfiles(var g : screenrecord;
                   var s : boolean);
var choice     : char;
    anyfile    : filename;
    choiceflag : boolean;
begin
  choiceflag := false;
  s := true;
  while not choiceflag do
  begin
    Write('What is Choice <1>,<2>,<3>,<C>,<Q> ? : ');
    readln(choice);
    writeln;

    case choice of
    '1' : begin
            g.usefile := false;
            g.filechoice.flow := none;
            choiceflag := true;
          end;
    '2' : begin
            g.usefile := true;
            g.filechoice.flow := input;
            Write('What is the Input file Name ?   : ');
            readln(g.filechoice.inputfilename);
            filetest(g.filechoice.inputfile,
                     g.filechoice.inputfilename,
                     1, choiceflag);
            writeln;
          end;

    '3' : begin
            g.usefile := true;
            g.filechoice.flow := output;
            Write('What is the Output file Name ? : ');
            readln(g.filechoice.outputfilename);
            filetest(g.filechoice.outputfile,
                     g.filechoice.outputfilename,
                     2, choiceflag);
            writeln;
          end;

     '4' : begin
```

```
9.16.  t := '';
  GetTime(h,m,s,hund);
  t := LeadingZero(h) + ':' + LeadingZero(m) + ':' + LeadingZero(s);
end;  { dotime }

procedure dodate(var t : numstr);
var   { get date from system }
  y, mo, d, dow : Word;
begin
  GetDate(y,mo,d,dow);
  t := leadingzero(d) + MONTHS[mo] +  leadingzero(y);
end;  { dotime }

procedure initialisegraphics(var gflag : boolean);
var       { check for a graphics driver and configure }
  grDriver: Integer;
  grMode  : Integer;
  ErrCode : Integer;
begin
  grDriver := Detect;
  InitGraph(grDriver, grMode,'c:\language\bp\bgi ');

  ErrCode := GraphResult;
  if ErrCode = grOk then
     gflag := true
  else
  begin
    Writeln('Graphics error:', GraphErrorMsg(ErrCode));
    gflag := false;
  end;    { end if errcode }
end; { initialisegraphics }

procedure printgraph(Mode : integer);
var i,j,k,   { loop counters }
    m,
    msb,lsb,Maxx,Maxy : integer;

begin
  maxx := getmaxx;
  maxy := getmaxy;
  { get whole screen }
  setviewport(0,0,maxx,maxy,false);

  write(lst, ESC,'A',#$07); { set line spacing to 7/72 inch }

  lsb := maxy AND $00ff;  { maxy mod 256 }
  msb := maxy shr 8;      { int maxx / 256 }

  j := 0;  { initial x value }
  repeat   { scan x values }
    write(lst,ESC,'*', char(mode),char(lsb),char(msb));
    for i := maxy downto 0 do    { y values to scan }
    begin
      m := 0;
      for k := 0 to 7 do
      begin    { construct m from individual pixel values }
        m := m shl 1; { shift m left 1 bit }
        if getpixel(j + k,i) <> 0 then inc(m); { if pixel not black }
      end;  { for k }                          { record it }
   write(lst, char(m));
  end; { for i }
```

```
    write(lst, #$0D, #$0A); { character 13 and character 10 }
    inc(j,8);
  until j >= maxx - 1;  { complete width used }
  write(lst,#$0C);  { form feed }
end;  { printgraph }


procedure dotext(gr : screenrecord;
                  i : integer);
var tint     : longint;
    temp     : word;
    tempstr,
    tmin,
    tavg,
    tmax     : numstr;
begin        {  display text screen data }
  settextjustify(lefttext,centertext);

  with gr do
  begin { set text viewport }
    initview(screen.t);

    setcolor(labelcolour);  { print headings }
    outtextxy(tcoll,trow,'MAX.RUNS');

    outtextxy(tcoll,trow *  3,'Run Number');

    outtextxy(tcoll,trow *  5,'TARGET');
    outtextxy(tcoll,trow *  6,'Radius');
    outtextxy(tcoll,trow *  7,'Area');

    outtextxy(tcoll,trow *  9,'BLUE');
    outtextxy(tcoll,trow * 10,'Count');
    outtextxy(tcoll,trow * 11,' Max');
    outtextxy(tcoll,trow * 12,' Avg');
    outtextxy(tcoll,trow * 13,' Min');

    outtextxy(tcoll,trow * 14,'Area');
    outtextxy(tcoll,trow * 15,' Max');
    outtextxy(tcoll,trow * 16,' Avg');
    outtextxy(tcoll,trow * 17,' Min');

    outtextxy(tcoll,trow * 18,'Density');
    outtextxy(tcoll,trow * 19,' Max');
    outtextxy(tcoll,trow * 20,' Avg');
    outtextxy(tcoll,trow * 21,' Min');
    outtextxy(tcoll,trow * 22,'Direction');
    outtextxy(tcoll,trow * 23,' Max');
    outtextxy(tcoll,trow * 24,' Avg');
    outtextxy(tcoll,trow * 25,' Min');

    outtextxy(tcoll,trow * 27,'GREEN AREA');
    outtextxy(tcoll,trow * 28,'Per Cent');

    outtextxy(tcoll,trow * 30,'Exp.Blue');
    outtextxy(tcoll,trow * 31,'ave %');

    outtextxy(tcoll,trow * 33,'Overlap');

    outtextxy(tcoll,trow * 34,'Total Bits');
    outtextxy(tcol1,trow * 35,'Files');    { use files }

    if gr.usefile then
      case gr.filechoice.flow of
      input  : begin
                 outtextxy(tcol1,trow * 36,' Input');
                 tempstr := gr.filechoice.inputfilename;
                 outtextxy(tcol1, trow * 37,tempstr);
               end;
      output : begin
                 outtextxy(tcol1,trow * 36,' Output');
                 tempstr := gr.filechoice.outputfilename;
                 outtextxy(tcol1, trow * 37,tempstr);
               end;
      inout  : begin
                 outtextxy(tcol1,trow * 36,' Input');
                 tempstr := gr.filechoice.inputfilename;
                 outtextxy(tcol1, trow * 37,tempstr);

                 outtextxy(tcol1,trow * 38,' Output');
                 tempstr := gr.filechoice.outputfilename;
                 outtextxy(tcol1, trow * 39,tempstr);
               end;
      end; { case }

  { overlapping }
    if i = gr.maxrun then
    begin
      outtextxy(tcol1,trow * 42,'R to repeat');
      outtextxy(tcol1,trow * 43,'Q to quit');
      outtextxy(tcol1,trow * 44,'P to print');
    end;  { if finish is true }
    outtextxy(tcol1,trow * 45,'Time');
    outtextxy(tcol1,trow * 46,'Date');

  { write in values }
    settextjustify(righttext,centertext);
    setcolor(textcolour);
    str(gr.maxrun,tempstr);
    outtextxy(tcol2,trow * 1,tempstr);
    str(gr.loop,tempstr);
    outtextxy(tcol2,trow * 3,tempstr);

    str(gr.tradius,tempstr);
    outtextxy(tcol2,trow *  6,tempstr);
    str(gr.targetarea,tempstr);
    outtextxy(tcol2,trow *  7,tempstr);

    str(gr.bluecount[counter],tempstr);
    str(gr.bluecount[max],tmax);
    str(gr.bluecount[avg],tavg);
    str(gr.bluecount[min],tmin);

    outtextxy(tcol2,trow * 10,tempstr);
    outtextxy(tcol2,trow * 11,tmax);
    outtextxy(tcol2,trow * 12,tavg);
    outtextxy(tcol2,trow * 13,tmin);

    str(gr.bluearea[counter],tempstr);
    str(gr.bluearea[max],tmax);
    str(gr.bluearea[avg],tavg);
    str(gr.bluearea[min],tmin);

    outtextxy(tcol2,trow * 14,tempstr);
    outtextxy(tcol2,trow * 15,tmax);
    outtextxy(tcol2,trow * 16,tavg);
    outtextxy(tcol2,trow * 17,tmin);

    str(gr.bluedensity[counter]:2:4,tempstr);
    str(gr.bluedensity[max]:2:4,tmax);
    str(gr.bluedensity[avg]:2:4,tavg);
    str(gr.bluedensity[min]:2:4,tmin);

    outtextxy(tcol2,trow * 18,tempstr);
    outtextxy(tcol2,trow * 19,tmax);
    outtextxy(tcol2,trow * 20,tavg);
    outtextxy(tcol2,trow * 21,tmin);

    str(0,tempstr);
    str(gr.bluedirection[max],tmax);
    str(gr.bluedirection[avg],tavg);
    str(gr.bluedirection[min],tmin);

    outtextxy(tcol2,trow * 22,tempstr);
    outtextxy(tcol2,trow * 23,tmax);
    outtextxy(tcol2,trow * 24,tavg);
    outtextxy(tcol2,trow * 25,tmin);

    str(gr.greenarea:2:2,tempstr);
    outtextxy(tcol2,trow * 28,tempstr);

    tempstr := num2str(gr.expectedblue);
    outtextxy(tcol2,trow * 30, tempstr);
    if gr.expectedblue <> 0 then
    begin
      gr.aveexpblue := (gr.bluecount[avg] / gr.expectedblue)*100;
      str(gr.aveexpblue:2:4,tempstr);
      outtextxy(tcol2,trow * 31,tempstr);
    end; { if }

    if gr.overlap then
      tempstr := 'Yes'
    else
      tempstr := 'No';

    outtextxy(tcol2,trow * 33, tempstr);

    tempstr := num2str(gr.totalbits);
    outtextxy(tcol2,trow * 34,tempstr);

    if not gr.usefile then outtextxy(tcol2, trow * 35,'none');

    outtextxy(tcol2,trow * 45, gr.timestr);
    outtextxy(tcol2,trow * 46, gr.datestr);

  end; { with  view }
end; { dotext }

procedure getcoords(x,y,r,deg : integer;
                         var t : pointtype );
begin     { calculate coordinates for radial labels }
  t.x := trunc(sin(deg * degree) * r);
  t.y := trunc(cos(deg * degree) * r);
  t.x := x + t.x;
  t.y := y + t.y;
end;  { getcoords }

procedure cross(x,y : integer);
var temp : integer;
begin  { draw a cross at coordinates }
  setcolor(crosscolour);
  temp := 5;
  line(x-temp,y,x+temp,y);
  Line(x,y-temp,x,y+temp);
  circle(x,y,temp -2);
end; { cross }

procedure dosquare2(var g: screenrecord);
var count : integer;
    coord : pointtype;
    increment : integer;
begin
  with g do
  begin
   { initview(screen.g);
   } cross(centre.x,centre.y);

    setcolor(gratcolour);
    rectangle(centre.x - tradius,centre.y - tradius,
              centre.x + tradius,centre.y + tradius); { print target }
    increment := tradius + 10;
    settextjustify(centertext,centertext);

    outtextxy(centre.x,            centre.y - increment,'00');
    outtextxy(centre.x + increment,centre.y,            '01');
    outtextxy(centre.x,            centre.y + increment,'10');
    outtextxy(centre.x - increment,centre.y,            '11');
  end; { with g }
end;  { dosquare2 }

procedure dosquare3(var g: screenrecord);
var coord     : pointtype;
    increment : integer;
begin
  with g do
  begin
    initview(screen.g);
    cross(centre.x,centre.y);
    { print target }
    setcolor(gratcolour);
    rectangle(centre.x - tradius,centre.y - tradius,
              centre.x + tradius,centre.y + tradius);
    increment := tradius + 15;

    settextjustify(centertext,centertext);
    outtextxy(centre.x,             centre.y - increment,'000');
    outtextxy(centre.x + increment, centre.y - increment,'001');
    outtextxy(centre.x + increment, centre.y            ,'010');
    outtextxy(centre.x + increment, centre.y + increment,'011');
    outtextxy(centre.x,             centre.y + increment,'100');
    outtextxy(centre.x - increment, centre.y + increment,'101');
    outtextxy(centre.x - increment, centre.y,            '110');
    outtextxy(centre.x - increment, centre.y   increment,'111');
  end; { with g }
end;  { dosquare3 }

procedure dosquare4(var g: screenrecord);
var coord : pointtype;
    halfinc,
    increment : integer;
begin
  with g do
  begin
    initview(screen.g);
    cross(centre.x,centre.y);

    setcolor(gratcolour);
    rectangle(centre.x - tradius,centre.y - tradius,
              centre.x + tradius,centre.y + tradius); { print target }
    increment := tradius + 25;
    halfinc := trunc(increment/2);

    settextjustify(centertext,centertext);

    outtextxy(centre.x,             centre.y - increment,'0000');
    outtextxy(centre.x + halfinc,   centre.y - increment,'0001');
    outtextxy(centre.x + increment, centre.y - increment,'0010');
    outtextxy(centre.x + increment, centre.y - halfinc,  '0011');
    outtextxy(centre.x + increment, centre.y,            '0100');
    outtextxy(centre.x + increment, centre.y + halfinc,  '0101');
    outtextxy(centre.x + increment, centre.y + increment,'0110');
    outtextxy(centre.x + halfinc,   centre.y + increment,'0111');
    outtextxy(centre.x,             centre.y + increment,'1000');
    outtextxy(centre.x - halfinc,   centre.y + increment,'1001');
    outtextxy(centre.x - increment, centre.y + increment,'1010');
    outtextxy(centre.x - increment, centre.y + halfinc,  '1011');
    outtextxy(centre.x - increment, centre.y,            '1100');
    outtextxy(centre.x - increment, centre.y - halfinc,  '1101');
    outtextxy(centre.x - increment, centre.y - increment,'1110');
    outtextxy(centre.x - halfinc,   centre.y - increment,'1111');
  end;  { with g }
end;  { dosquare4 }

procedure dotarget2(var g : screenrecord);
var coord : pointtype;
begin           { do 2 bit target }
  with g do
  begin
    initview(screen.g);
    cross(centre.x,centre.y);

    setcolor(gratcolour);
    circle(centre.x,centre.y,tradius); { print target }

    settextjustify(centertext,centertext);

    getcoords(centre.x, centre.y, tradius + 10,180, coord);
    outtextxy(coord.x,coord.y,'00');
    getcoords(centre.x, centre.y, tradius + 20, 90, coord);
    outtextxy(coord.x,coord.y,'01');
    getcoords(centre.x, centre.y, tradius + 10,  0, coord);
    outtextxy(coord.x,coord.y,'10');
    getcoords(centre.x, centre.y, tradius + 20,270, coord);
    outtextxy(coord.x,coord.y,'11');
  end;  { with g }
end; { dotarget2 }

procedure dotarget3(var g : screenrecord);
var coord : pointtype;
begin
  with g do
  begin
    initview(screen.g);
    cross(g.centre.x,g.centre.y);

    setcolor(gratcolour);   { draw target }
       { double thickness to eliminate bleed through }
    circle(g.centre.x,g.centre.y, g.tradius);
    circle(g.centre.x,g.centre.y, g.tradius+1);

    settextjustify(centertext,centertext);

    getcoords(g.centre.x, g.centre.y, g.tradius +10,180, coord);
    outtextxy(coord.x,coord.y,'000');
    getcoords(g.centre.x, g.centre.y, g.tradius +20,135, coord);
    outtextxy(coord.x,coord.y,'001');
    getcoords(g.centre.x, g.centre.y, g.tradius +20,90, coord);
    outtextxy(coord.x,coord.y,'010');
    getcoords(g.centre.x, g.centre.y, g.tradius +20,45, coord);
    outtextxy(coord.x,coord.y,'011');
    getcoords(g.centre.x, g.centre.y, g.tradius +10,0, coord);
    outtextxy(coord.x,coord.y,'100');
    getcoords(g.centre.x, g.centre.y, g.tradius +20,315, coord);
    outtextxy(coord.x,coord.y,'101');
    getcoords(g.centre.x, g.centre.y, g.tradius +20,270, coord);
    outtextxy(coord.x,coord.y,'110');
    getcoords(g.centre.x, g.centre.y, g.tradius +20,225, coord);
    outtextxy(coord.x,coord.y,'111');
  end;  { with g }
end; { dotarget3 }

procedure dotarget4(var g : screenrecord);
var coord : pointtype;
begin
  with g do
  begin
    initview(screen.g);
    cross(g.centre.x,g.centre.y);

    setcolor(gratcolour);   { draw target }
       { double thickness to eliminate bleed through }
    circle(g.centre.x,g.centre.y, g.tradius);
    circle(g.centre.x,g.centre.y, g.tradius+1);
    circle(g.centre.x,g.centre.y, g.tradius+2);

    settextjustify(centertext,centertext);

    getcoords(g.centre.x, g.centre.y, g.tradius + 10, 180, coord);
    outtextxy(coord.x,coord.y,'0000');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 157, coord);
    outtextxy(coord.x,coord.y,'0001');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 135, coord);
    outtextxy(coord.x,coord.y,'0010');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 112, coord);
    outtextxy(coord.x,coord.y,'0011');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20,  90, coord);
    outtextxy(coord.x,coord.y,'0100');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20,  68, coord);
    outtextxy(coord.x,coord.y,'0101');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20,  45, coord);
    outtextxy(coord.x,coord.y,'0110');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20,  22, coord);
    outtextxy(coord.x,coord.y,'0111');
    getcoords(g.centre.x, g.centre.y, g.tradius + 10,   0, coord);
    outtextxy(coord.x,coord.y,'1000');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 338, coord);
    outtextxy(coord.x,coord.y,'1001');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 315, coord);
    outtextxy(coord.x,coord.y,'1010');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 293, coord);
    outtextxy(coord.x,coord.y,'1011');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 270, coord);
    outtextxy(coord.x,coord.y,'1100');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 248, coord);
    outtextxy(coord.x,coord.y,'1101');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 225, coord);
    outtextxy(coord.x,coord.y,'1110');
    getcoords(g.centre.x, g.centre.y, g.tradius + 20, 202, coord);
    outtextxy(coord.x,coord.y,'1111');
  end; { with g }
end; { dotarget4 }

procedure dograticule(var gr : screenrecord);
var xlimit, ylimit : integer;
begin
  xlimit := getmaxx; { get screen limits }
  ylimit := getmaxy;
  setcolor(linecolour);
  rectangle(0,0,xlimit, ylimit);          { set graticule }
  rectangle(3,3,xlimit - 3,ylimit - 3);
  line(479,3,479,ylimit - 3);
  with gr do
  begin
    setviewport(479,4,479,ylimit-3,clipon); { set text area }
    getviewsettings(screen.t.port); { store text area limits }
    screen.t.fill := lightgray;

    getviewsettings(screen.h.port); { store help screen limits }
    screen.h.fill := yellow;

    setviewport(4,4,478,478, clipon);   { graphics area }

    getviewsettings(screen.g.port);  { store graphics area limits }
    screen.g.fill := white;

    getviewsettings(screen.i.port);  { store data input area limits }
    screen.i.fill := blue;
  end;  { with gr }
end; { dograticule }


procedure initscreenrec(var g : screenrecord);
begin

  with g do
  begin
```

```
    with screen.g do
    begin
      centre.x := (port.x2 - port.x1) div 2;
      centre.y := (port.y2 - port.y1) div 2;
      fill := gbackcolour;
    end; { with }
    tradius := midradius;  { default settings }
    maxrun  := maxruns;
    shape   := 1;
    step    := 1;

    overlap := false;
    numberbits := 2;   { default two bit patterns }

    expectedblue := tradius * tradius;

    aveexpblue   := 0;

    totalbits    := 0;
    bluerunarea  := 0;
    greenarea    := 0;
    bluerunnumber  := 0;
    bluerundensity := 0;

    bluearea[min]      := maxlongint;
    bluedensity[min]   := maxlongint;
    bluedirection[min] := maxlongint;
    bluecount[min]     := maxlongint;


    dotime(timestr);
    dodate(datestr);
  end;

end; { initscreenrec }


procedure dohelp(t : view);

var temp : integer;
begin              { print help data on screen }
  settextjustify(lefttext,centertext);

  with t do
  begin
    initview(t);

    setcolor(labelcolour);
    outtextxy(tcol1,trow,'Help Screen');
    outtextxy(tcol1,trow *  3,'Run :');
    outtextxy(tcol1,trow *  6,'Index :');
    outtextxy(tcol1,trow *  9,'Density :');
    outtextxy(tcol1,trow * 12,'Direction:');
    outtextxy(tcol1,trow * 15,'COUNT');
    outtextxy(tcol1,trow * 17,'MAX');
    outtextxy(tcol1,trow * 19,'AVG');
    outtextxy(tcol1,trow * 21,'MIN');
    outtextxy(tcol1,trow * 34,'Target(in Pixels)');
    outtextxy(tcol1,trow * 36,'SIZE :');
    outtextxy(tcol1,trow * 38,'AREA :');
    setcolor(textcolour);
    outtextxy(tcol1,trow *  4,'Number of Runs');
    outtextxy(tcol1,trow *  7,'Number of counts');
    outtextxy(tcol1,trow * 10,'No. of Blues');
    outtextxy(tcol1,trow * 13,'Avg. Direction');
    outtextxy(tcol1,trow * 16,'total this run');
    outtextxy(tcol1,trow * 18,'Max. of all runs');
    outtextxy(tcol1,trow * 20,'Avg. of all runs');
    outtextxy(tcol1,trow * 22,'Min. of all runs');
    outtextxy(tcol1,trow * 36,'         radius');
    outtextxy(tcol1,trow * 38,'         pixels');

    outtextxy(tcol1,trow * 40,'KEY COMMANDS');
    outtextxy(tcol1,trow * 44,'P  : Print');
    outtextxy(tcol1,trow * 46,'Q  : Quit');
  end;  { with h }
  { return to graphics window }
end; { dohelp }

procedure doinput(var gr : screenrecord);
var pos,
    temp,
    code,
    tsize,
    truns : integer;
    tstr1,
    tstr2 : numstr;
    tch : char;
    templstr,
    longblank : string[40];
    inputstr : string[6];
    overlap : boolean;
begin          { change session parameters }
  settextjustify(lefttext,centertext);

  inputstr := '';
  with gr do
  begin
    initview(screen.i);
    setcolor(labelcolour);

    outtextxy(tcol1,trow,'Data Input Screen');

    tstr1 := num2str(minradius);
    tstr2 := num2str(maxradius);
    outtextxy(tcol1,trow * 5,'<T>arget Radius Min. ' + tstr1
                         + ' Max.' + tstr2);
    outtextxy(tcol1,trow * 6,'Current Radius');
    tstr1 := num2str(gr.tradius);
    outtextxy(tcol2,trow * 6,tstr1);

    outtextxy(tcol1,trow * 8,'<O>verlap');
    if gr.overlap then outtextxy(tcol2,trow * 8,'yes')
    else outtextxy(tcol2,trow * 8, 'no');

    outtextxy(tcol1,trow * 10,'Number of <R>uns Min : 1  Max : 32767');
    outtextxy(tcol1,trow * 11,'Current Runs');
    tstr1 := num2str(gr.maxrun);
    outtextxy(tcol2,trow * 11,tstr1);

    tstr1 := num2str(gr.numberbits);
```

```
outtextxy(tcol1,trow * 13,'Number of <B>its ');
outtextxy(tcol2,trow * 13,tstr1);

outtextxy(tcol1,trow * 15,'<S>hape Default ');
if gr.shape = 1 then  outtextxy(tcol2,trow * 15,'Square')
else outtextxy(tcol2,trow * 15,'Circle');

outtextxy(tcol1,trow * 22,'Select <?> letter to change values');
outtextxy(tcol1,trow * 23,'Press return after entering each value');

outtextxy(tcol1,trow * 25,'EXIT this screen : left arrow');

setcolor(textcolour);
while not keypressed do
begin
  tch := readkey;
  if tch = #0 then
  begin
    tch := readkey;
    case tch of
    'K' : begin    { exit - left arrow }
            pos := 25;
            setcolor(blankcolour);
            outtextxy(tcol1,trow * pos, blank6str);
            setcolor(changecolour);
            outtextxy(tcol1,trow * pos,'EXIT');
            break;
          end;
     end; { case }
  end
  else { not escape character }
  begin
    case tch of
    'B','b'  : begin  { change number of bits }
                 inc(gr.numberbits);
                 if gr.numberbits > 4 then gr.numberbits := 2;

                 setcolor(blankcolour);
                 outtextxy(tcol2,trow * 13, blank6str);
                 setcolor(changecolour);
                 outtextxy(tcol2,trow * 13, num2str(gr.numberbits));
               end;
    'S','s'  : begin    { select square or circle target }
                 setcolor(blankcolour);
                 outtextxy(tcol2,trow * 15,blank6str);
                 setcolor(changecolour);
                 if gr.shape = 1 then
                 begin
                   gr.shape := 2;
                   outtextxy(tcol2,trow * 15,'Circle');
                 end
                 else
                 begin
                   gr.shape := 1;
                   outtextxy(tcol2,trow * 15,'Square');
                 end;
               end;
     'T','t' : begin   { change size of target }
                 pos := 6;
                 setcolor(blankcolour);
                 outtextxy(tcol2,trow * pos, blank6str);
```

```
                     setcolor(changecolour);
                     tsize := getnum(inputstr,pos,tcol2);
                     if tsize > maxradius then tsize := maxradius;
                     if tsize < 50 then tsize := 50;
                     setcolor(blankcolour);
                     outtextxy(tcol2,trow * pos, blank6str);
                     setcolor(changecolour);
                     outtextxy(tcol2,trow * pos, num2str(tsize));
                     gr.tradius := tsize;
                     gr.expectedblue := tsize * tsize;
                   end;
         'O','o' : begin  { use overlapping samples }
                     pos  := 8;
                     setcolor(blankcolour);
                     outtextxy(tcol2,trow * pos, blank6str);
                     overlap := not overlap;
                     setcolor(changecolour);
                     if overlap then
                       outtextxy(tcol2,trow * pos,'yes')
                     else
                       outtextxy(tcol2,trow * pos,'no');

                     gr.overlap := overlap;
                   end;
         'R','r' : begin   { change number of runs }
                     pos := 11;
                     setcolor(blankcolour);
                     outtextxy(tcol2,trow * pos, blank6str);
                     setcolor(changecolour);

                     truns := getnum(inputstr,pos,tcol2);
                     if truns > maxint then truns := maxint;

                     gr.maxrun := truns;
                   end;
         'H','h' : dohelp(screen.h);
         end; { case }

      end; {if tch }

      setcolor(labelcolour);
    end; { while not keypressed }

    tch := 'x';
  end;  { with i }
    clearviewport;
end;  {  doinput }

procedure getbit(var f : sintfile;
                 var b : shortint);
begin
  read(f,b);

end; { get bit }
procedure putbit(var f : sintfile;
                 var b : shortint);
begin
  write(f,b);
end; { putbit }
```

```
procedure density(var g : screenrecord);
var indexx, indexy : integer;
    total1,
    total2 : longint;

begin  { count number of tracecolour and areacolour pixels }
  total1 := 0;
  total2 := 0;
  setcolor(areacolour);
  with g do
  begin
  for indexx := (centre.x - (tradius + 1))
   to (centre.x + (tradius + 1)) do
    for indexy := (centre.y - (tradius + 1))
     to (centre.y + (tradius + 1)) do
    begin
      pixelcolor := getpixel(indexx,indexy);
      case pixelcolor of
       tracecolour : begin   { turn trace to green }
                       putpixel(indexx,indexy,areacolour);
                       inc(total1);
                       inc(total2);
                     end;
       areacolour : begin    { count total area }
                     inc(total2);
                   end;
      end; { case }
    end; { for }
     bluecount[counter] := bluerunnumber;
     bluerunarea := total1;
     greenarea := total2;

     bluearea[counter] := bluerunarea;

     bluerundensity := (bluecount[counter]/bluearea[counter]);
     bluedensity[counter] := bluerundensity;
     greenarea := (greenarea/targetarea) * 100;

     doirec(bluecount,loop);      { store data in a record }
     doirec(bluearea,loop);
     dorrec(bluedensity,loop);
     doirec(bluedirection,loop);

  end; { with g }

end;  { density }



procedure dorun2(var g : screenrecord;l : longint);
var oldcolour : word;
r2 : twoarray;
count : shortint;

begin    { generate and plot pixels on target in 2 bits }
{ if files are used }

  if g.usefile and ((g.filechoice.flow = input)
              or (g.filechoice.flow = inout)) then
  begin
    getbit(g.filechoice.inputfile, r2[1]);
    getbit(g.filechoice.inputfile, r2[2]);
  end
  else
  begin
    for count := 1 to 2 do  { intitialise array }
      r2[count] := random(2);  { get random value }
  end;

  if g.usefile and (g.filechoice.flow = output)
               or (g.filechoice.flow = inout) then
  begin
    putbit(g.filechoice.outputfile,r2[1]);
    putbit(g.filechoice.outputfile,r2[2]);
  end; { if }

  setcolor(tracecolour);

  while not restart do
  begin
    inc(1);

    if r2[1] = 0 then
      case r2[2] of
          0 : begin     { 00 }
                next.x := current.x ;     { up }
                next.y := current.y - 1;
              end;
          1 : begin     { 01 }
                next.x := current.x + 1;
                next.y := current.y;
              end;
      end;  { caser2[1] = 0 }

    if r2[1] = 1 then
      case r2[2] of
          0 : begin     { 10 }
                next.x := current.x;      { down }
                next.y := current.y + 1;
              end;
          1 : begin    { 11}
                next.x := current.x - 1;  { left }
                next.y := current.y;
              end;
      end; { case }

    { write data to file }

    if g.overlap then
    begin
      r2[1] := r2[2]; { swap }

      if g.usefile and (g.filechoice.flow = input)
                   or (g.filechoice.flow = inout) then
        getbit(g.filechoice.inputfile,r2[2])
      else
        r2[2] := random(2);

        if (g.filechoice.flow = output) or (g.filechoice.flow = inout)
then
          putbit(g.filechoice.outputfile,r2[2]);
        inc(g.totalbits)
    end; { if overlap }

    if not g.overlap then
    begin
      if g.usefile and (g.filechoice.flow = input)
                   or (g.filechoice.flow = inout) then
      begin
        getbit(g.filechoice.inputfile,r2[1]);
        getbit(g.filechoice.inputfile,r2[2]);

      end
      else
      begin
        r2[1] := random(2);
        r2[2] := random(2);
      end; { if }
      if g.usefile and (g.filechoice.flow = output)
                   or (g.filechoice.flow = inout) then
      begin
        putbit(g.filechoice.outputfile,r2[1]);
        putbit(g.filechoice.outputfile,r2[2]);
      end; { if }
      inc(g.totalbits,2);
    end;


    pixelcolor := getpixel(next.x,next.y);

    if pixelcolor = gratcolour then
    begin      { if hit target rim }
      restart := true;
    end;

    putpixel(next.x,next.y,tracecolour); { illuminate point }
    current := next;

    ch := 'x';

  end; { while not finish }
  g.bluerunnumber := 1;
  cross(next.x,next.y);

end;  { dorun2 }

procedure dorun3(var g : screenrecord;l : longint);
var oldcolour : word;
    count : integer;
    r3 : threearray;
begin
  if g.usefile and (g.filechoice.flow = input)
               or (g.filechoice.flow = inout) then
  begin
    getbit(g.filechoice.inputfile, r3[1]);
    getbit(g.filechoice.inputfile, r3[2]);
    getbit(g.filechoice.inputfile, r3[3]);
  end
  else
  begin
    for count := 1 to 3 do    { intitialise array }
      r3[count] := random(2);  { get random value }
```

```
end;

if g.usefile and (g.filechoice.flow = output)
             or (g.filechoice.flow = inout) then
begin
  putbit(g.filechoice.outputfile,r3[1]);
  putbit(g.filechoice.outputfile,r3[2]);
  putbit(g.filechoice.outputfile,r3[3]);

end; { if }


setcolor(tracecolour);
while not restart do
begin
  inc(l);

  case r3[1] of

    0 : begin
          case r3[2] of

            0 : begin
                  case r3[3] of
                    0 : begin    { up 000 }
                          next.x := current.x ;
                          next.y := current.y - 1;
                        end;
                    1 : begin    { up and right 001 }
                          next.x := current.x + 1;
                          next.y := current.y - 1;
                        end;
                  end; { case r3[3] }
                end;
            1 : begin
                  case r3[3] of
                    0 : begin    { right 010 }
                          next.x := current.x + 1;
                          next.y := current.y;
                        end;
                    1 : begin    { right and down 011 }
                          next.x := current.x + 1;
                          next.y := current.y + 1;
                        end;
                    end { case r3[3] }
                end; { case r3[2] }
           end; {case r3[2] }
        end; { case r3[1] = 0 }

    1 : begin
            case r3[2] of

            0 : begin
                  case r3[3] of
                    0 : begin    { down 100 }
                          next.x := current.x ;
                          next.y := current.y + 1;
                        end;
                    1 : begin    { down and left 101 }
                          next.x := current.x - 1;
                          next.y := current.y + 1;
                          end;
                      end; { case r3[3] }
                  end;
              1 : begin
                    case r3[3] of
                      0 : begin    { left 110 }
                            next.x := current.x - 1;
                            next.y := current.y;
                          end;
                      1 : begin    { left and up 111 }
                            next.x := current.x - 1;
                            next.y := current.y - 1;
                          end;
                      end { case r3[3] }
                  end; { case r3[2] = 1 }
              end; {case r3[2] }
          end; { case r3[1] = 1 }

      end; { case r3 }

     if g.overlap then
     begin

       r3[1] := r3[2]; { store previous values for next run }
       r3[2] := r3[3];
      if g.usefile and (g.filechoice.flow = input)
                   or (g.filechoice.flow = inout) then
        getbit(g.filechoice.inputfile,r3[3])
      else
        r3[3] := random(2);

      if (g.filechoice.flow = output) or (g.filechoice.flow = inout)
then
          putbit(g.filechoice.outputfile,r3[3]);

       inc(g.totalbits);
     end     { get next value }
     else  { non-overlap }
     begin
        if g.usefile and (g.filechoice.flow = input)
                     or (g.filechoice.flow = inout) then
        begin
          getbit(g.filechoice.inputfile,r3[1]);
          getbit(g.filechoice.inputfile,r3[2]);
          getbit(g.filechoice.inputfile,r3[3]);
        end
        else
        begin
          for count := 1 to 3 do   { refill array }
            r3[count] := random(2);
        end;
        if g.usefile and (g.filechoice.flow = output)
                     or (g.filechoice.flow = inout) then
        begin
          putbit(g.filechoice.outputfile,r3[1]);
          putbit(g.filechoice.outputfile,r3[2]);
          putbit(g.filechoice.outputfile,r3[3]);
        end;
       inc(g.totalbits,3);
     end;
```

```
    pixelcolor := getpixel(next.x,next.y);

    if pixelcolor = gratcolour then
    begin
      restart := true;
    end;

    putpixel(next.x,next.y,tracecolour);
    current := next;

    ch := 'x';

  end; { while not finish }
  g.bluerunnumber := 1;
  cross(next.x,next.y);
end;  { dorun3 }

procedure dorun4(var g : screenrecord;l : longint);
var oldcolour : word;
    index,
    count     : integer;
    r4        : fourarray;
begin
  if g.usefile and (g.filechoice.flow = input)
                or (g.filechoice.flow = inout) then
  begin
    getbit(g.filechoice.inputfile, r4[1]);
    getbit(g.filechoice.inputfile, r4[2]);
    getbit(g.filechoice.inputfile, r4[3]);
    getbit(g.filechoice.inputfile, r4[4]);
  end
  else
  begin
    for count := 1 to 4 do   { intitialise array }
      r4[count] := random(2);  { get random value }
  end;

  if (g.filechoice.flow = output) or (g.filechoice.flow = inout) then
  begin
    putbit(g.filechoice.outputfile,r4[1]);
    putbit(g.filechoice.outputfile,r4[2]);
    putbit(g.filechoice.outputfile,r4[3]);
    putbit(g.filechoice.outputfile,r4[4]);
  end; { if }

  setcolor(tracecolour);
  while not restart do
  begin
    inc(l);

    case r4[1] of

    0 : begin
          case r4[2] of
          0 : begin
                case r4[3] of
                0 : begin
                      case r4[4] of
                      0 : begin
                            next.x := current.x ;
                            next.y := current.y - 2;
                      end;
                  1 : begin
                        next.x := current.x + 1;
                        next.y := current.y - 2;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end;
            1 : begin
                  case r4[4] of
                  0 : begin
                        next.x := current.x + 2;
                        next.y := current.y - 2;
                      end;
                  1 : begin
                        next.x := current.x + 2;
                        next.y := current.y - 1;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end; { r4[3] = 1 }
            end; { case r4[3] }
          end;  { r4[2] = 0 }
      1 : begin
            case r4[3] of
            0 : begin
                  case r4[4] of
                  0 : begin
                        next.x := current.x + 2;
                        next.y := current.y;
                      end;
                  1 : begin
                        next.x := current.x + 2;
                        next.y := current.y + 1;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end;
            1 : begin
                  case r4[4] of
                  0 : begin
                        next.x := current.x + 2;
                        next.y := current.y + 2;
                      end;
                  1 : begin
                        next.x := current.x + 1;
                        next.y := current.y + 2;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end; { r4[3] = 1 }
            end; { case r4[3] }

          end;  { r4[2] = 1 }
      end; { case r4[2] }
    end;  { r4[1] = 0 }
1 : begin
      case r4[2] of
      0 : begin
            case r4[3] of
            0 : begin
                  case r4[4] of
                  0 : begin
                        next.x := current.x;
                        next.y := current.y + 2;
                      end;
                  1 : begin
                        next.x := current.x - 1;
                        next.y := current.y + 2;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end;
            1 : begin
                  case r4[4] of
                  0 : begin
                        next.x := current.x - 2;
                        next.y := current.y + 2;
                      end;
                  1 : begin
                        next.x := current.x - 2;
                        next.y := current.y + 1;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end; { r4[3] = 1 }
            end; { case r4[3] }
          end;  { r4[2] = 0 }
      1 : begin
            case r4[3] of
            0 : begin
                  case r4[4] of
                  0 : begin
                        next.x := current.x - 2;
                        next.y := current.y ;
                      end;
                  1 : begin
                        next.x := current.x - 2;
                        next.y := current.y - 1;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end;
            1 : begin
                  case r4[4] of
                  0 : begin
                        next.x := current.x - 2;
                        next.y := current.y - 2;
                      end;
                  1 : begin
                        next.x := current.x - 1;
                        next.y := current.y - 2;
                      end;  { r4[4] = 1 }
                  end; { case r4[4] }
                end; { r4[3] = 1 }
            end; { case r4[3] }

          end;  { r4[2] = 1 }
      end; { case r4[2] }

    end;  { r4[1] = 1 }
end; { case r4[1]}


 if g.overlap then
 begin

   r4[1] := r4[2]; { store previous values for next run }
   r4[2] := r4[3];
```

```
       r4[3] := r4[4];

      if g.usefile and (g.filechoice.flow = input)
                   or (g.filechoice.flow = inout) then
        getbit(g.filechoice.inputfile,r4[4])
      else
        r4[4] := random(2);

      if (g.filechoice.flow = output) or (g.filechoice.flow = inout)
then
          putbit(g.filechoice.outputfile,r4[4]);

       inc(g.totalbits);
     end     { get next value }
     else  { non-overlap }
     begin
        if g.usefile and (g.filechoice.flow = input)
                     or (g.filechoice.flow = inout) then
        begin
          getbit(g.filechoice.inputfile,r4[1]);
          getbit(g.filechoice.inputfile,r4[2]);
          getbit(g.filechoice.inputfile,r4[3]);
          getbit(g.filechoice.inputfile,r4[4]);

        end
        else
        begin
          for count := 1 to 4 do   { refill array }
            r4[count] := random(2);
        end;
        if g.usefile and (g.filechoice.flow = output)
                     or (g.filechoice.flow = inout) then
        begin
          putbit(g.filechoice.outputfile,r4[1]);
          putbit(g.filechoice.outputfile,r4[2]);
          putbit(g.filechoice.outputfile,r4[3]);
          putbit(g.filechoice.outputfile,r4[4]);
        end;
       inc(g.totalbits,4);
     end;

    pixelcolor := getpixel(next.x,next.y);

    if pixelcolor = gratcolour then
    begin
      restart := true;
    end;

    putpixel(next.x,next.y,tracecolour);
    line(current.x,current.y, next.x,next.y);

    current := next;

    ch := 'x';

  end; { while not finish }
  g.bluerunnumber := 1;
  cross(next.x,next.y);
end;  { dorun4 }
```

```
Simulation of Test-T
program ttest;
{
  A program to perform the T Test on random sequences
  test to T4
}
uses crt;
const fore = 2;
      aft  = 8;
      numtests = 4;

      lowcolour  = lightcyan;
      expcolour  = white;
      highcolour = lightred;
      textcolour = yellow;
      backcolour = blue;

type
    onearray   = array[0..1]  of longint; { no of 0s and 1's }
    twoarray   = array[0..3]  of longint;
    threearray = array[0..7]  of longint;
    fourarray  = array[0..15] of longint;

    testarray = array[1..numtests] of shortint;

    devarray  = array[1..numtests} of real;
    anyfilename  = string[40];
var
  resultfilename : anyfilename;

  outfile : text;

  one   : onearray;    { accumulators to store values in }
  two   : twoarray;
  three : threearray;
  four  : fourarray;

  dev   : devarray;
  tests : testarray;

  testmax,
  count : longint;

  value : integer;
  zc    : real;
{____________________________________________________________}


procedure continue;
begin
  Write('Press enter to continue ');
  readln;
end; { continue }


procedure shuffle(var a : testarray;
                      v : integer);
var lcount : integer;
begin
  for lcount := 1 to 3 do
    a[lcount] := a[lcount + 1];
  a[4] := v;
end; { shuffle }

procedure docount1(var t : testarray;
                   var o : onearray);
begin
  case t[1] of
    0 : inc(o[0]);
    1 : inc(o[1]);
  end; { case }
end;  { docount1 }


procedure docount2(var t : testarray;
                   var o : twoarray);
begin

  case t[1] of
    0 : case t[2] of
          0 : inc(o[0]);
          1 : inc(o[1]);
        end; { case }
    1 : case t[2] of
          0 : inc(o[2]);
          1 : inc(o[3]);
        end; { case }
  end; { case }

end;  { docount2 }


procedure docount3(var t : testarray;
                   var o : threearray);
begin
  case t[1] of
    0 : case t[2] of
          0 : case t[3] of
                0 : inc(o[0]);
                1 : inc(o[1]);
              end;
          1 : case t[3] of
                0 : inc(o[2]);
                1 : inc(o[3]);
              end;
         end; { case t[2] }
    1 : case t[2] of
          0 : case t[3] of
                0 : inc(o[4]);
                1 : inc(o[5]);
              end;
          1 : case t[3] of
                0 : inc(o[6]);
                1 : inc(o[7]);
              end;
        end; { case t[2] }
    end; { end case t[1] }
end;  { docount3 }

procedure docount4(var t : testarray;
                   var o : fourarray);
begin
```

```
  case t[1] of
    0 : case t[2] of
          0 : case t[3] of
                0 : case t[4] of
                      0 : inc(o[0]);
                      1 : inc(o[1]);
                    end; { case }
                1 : case t[4] of
                      0 : inc(o[2]);
                      1 : inc(o[3]);
                    end; { case }
              end;
          1 : case t[3] of
                0 : case t[4] of
                      0 : inc(o[4]);
                      1 : inc(o[5]);
                    end; { case }
                1 : case t[4] of
                      0 : inc(o[6]);
                      1 : inc(o[7]);
                    end; { case }
              end;
         end; { case t[2] }
    1 : case t[2] of
          0 : case t[3] of
                0 : case t[4] of
                      0 : inc(o[8]);
                      1 : inc(o[9]);
                    end; { case }
                1 : case t[4] of
                      0 : inc(o[10]);
                      1 : inc(o[11]);
                    end; { case }
              end;
          1 : case t[3] of
                0 : case t[4] of
                      0 : inc(o[12]);
                      1 : inc(o[13]);
                    end; { case }
                1 : case t[4] of
                      0 : inc(o[14]);
                      1 : inc(o[15]);
                    end; { case }
              end;
         end; { case t[2] }


    end; { end case t[1] }
end;  { docount4 }

procedure displayarray(t : testarray);
var lcount : integer;
begin
  for lcount := 1 to 4 do
    write(t[lcount]);
  writeln;
end; { displayarray }

procedure choosecolour(var r,h,l : real);
begin
```

```
  if r > h then textcolor(lowcolour);
  if (r <= h ) and (r>= l) then textcolor(expcolour);
  if r < l then textcolor(highcolour);
end; { shoosecolour }

procedure doheader(l,e,h : real);
begin
  textcolor(lowcolour);
  write('Low ',l:fore:aft, '         ');
  textcolor(expcolour);
  write(' exp. ',e:fore:aft,'         ');
  textcolor(highcolour);
  writeln(' high ',h:fore:aft);
end; { doheader }

procedure getresult1(o : onearray;
                     t : longint;
                     z : real;
                     var f : text );
var lresult, lexpected, high,low : real;
begin
  lexpected := 1/2;
  low := lexpected - z;
  high := lexpected + z;

  doheader(low,lexpected,high);

  lresult := o[0]/t;
  choosecolour(lresult,high,low);
  write(' ':12,'0s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[1]/t;
  choosecolour(lresult,high,low);
  writeln('   1s : ',lresult:fore:aft);
  writeln(f,lresult);
  textcolor(yellow);
end; { getresult1 }

procedure getresult2(o : twoarray;
                     t : longint;
                     z : real;
                     var f :  text );
var lresult, lexpected, low,high : real;
begin
  lexpected := 1/4;
  low := lexpected - z;
  high := lexpected + z;
  doheader(low,lexpected,high);
  lresult := o[0]/t;
  choosecolour(lresult,high,low);
  write(' ':11,'00s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[1]/t;
  choosecolour(lresult,high,low);
  writeln('  01s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[2]/t;
  choosecolour(lresult,high,low);
  write(' ':11,'10s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[3]/t;
  choosecolour(lresult,high,low);
  writeln('  11s : ',lresult:fore:aft);
  writeln(f,lresult);
  textcolor(yellow);
end; { getresult2 }

procedure getresult3(o : threearray;
                     t : longint;
                     z : real;
                     var f : text);
var lresult, lexpected, low,high : real;
begin
  lexpected := 1/8;
  low := lexpected - z;
  high := lexpected + z;

  doheader(low,lexpected,high);

  lresult := o[0]/t;
  choosecolour(lresult,high,low);
  write(' ':10,'000s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[1]/t;
  choosecolour(lresult,high,low);
  writeln(' 001s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[2]/t;
  choosecolour(lresult,high,low);
  write(' ':10,'010s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[3]/t;
  choosecolour(lresult,high,low);
  writeln(' 011s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[4]/t;
  choosecolour(lresult,high,low);
  write(' ':10,'100s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[5]/t;
  choosecolour(lresult,high,low);
  writeln(' 101s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[6]/t;
  choosecolour(lresult,high,low);
  write(' ':10,'110s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[7]/t;
  choosecolour(lresult,high,low);
  writeln(' 111s : ',lresult:fore:aft);
  writeln(f,lresult);
  textcolor(yellow);
end; { getresult3 }

procedure getresult4(o : fourarray;
                     t : longint;
                     z : real;
                     var f : text);

var lresult, lexpected,low,high : real;
begin
  lexpected := 1/16;
  low := lexpected - z;
  high := lexpected + z;


  doheader(low,lexpected,high);
  lresult := o[0]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'0000s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[1]/t;
  choosecolour(lresult,high,low);
  writeln('0001s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[2]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'0010s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[3]/t;
  choosecolour(lresult,high,low);
  writeln('0011s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[4]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'0100s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[5]/t;
  choosecolour(lresult,high,low);
  writeln('0101s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[6]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'0110s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[7]/t;
  choosecolour(lresult,high,low);
  writeln('0111s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[8]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'1000s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);
  lresult := o[9]/t;
  choosecolour(lresult,high,low);
  writeln('1001s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[10]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'1010s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[11]/t;
  choosecolour(lresult,high,low);
  writeln('1011s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[12]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'1100s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[13]/t;
  choosecolour(lresult,high,low);
  writeln('1101s : ',lresult:fore:aft);
  writeln(f,lresult);

  lresult := o[14]/t;
  choosecolour(lresult,high,low);
  write(' ':9,'1110s : ',lresult:fore:aft,'    ');
  writeln(f,lresult);

  lresult := o[15]/t;
  choosecolour(lresult,high,low);
  writeln('1111s : ',lresult:fore:aft);
  writeln(f,lresult);
  textcolor(yellow);
end; { getresult4 }

function getZ(z,p : real;
              n   : longint): real;
var temp : real;
begin
  temp := z * sqrt(P * (1 - p)/n);
  getZ := temp;
end; { getZ }

procedure dodev(var a : devarray;
                    z : real;
                    n : longint);
var count : integer;
begin
  for count := 1 to numtests do
   case count of
   1 :  a[count] := getz(Z,0.5,n);
   2 :  a[count] := getz(Z,0.25,n);
   3 :  a[count] := getz(Z,0.125,n);
   4 :  a[count] := getz(Z,0.0625,n);
   end;  { case }
end; { dodev }

{-----------------------------------------------------}
begin
```

```
  textbackground(backcolour);  { clear Screen }
  textcolor(textcolour);
  clrscr;
                               { display header }
  writeln(' ':20,'Simulation of Test-T for 4 Bits');
  writeln;



  write('What is the name of the File to store results in ? : ');
  readln(resultfilename);

  if resultfilename = ''then  resultfilename := 'dummy.dta';

  assign(outfile,resultfilename);
  rewrite(outfile);

  write('What is the Standard Deviation value ?              : ');
  readln(zc);

  write('What is the number of bits to be tested ?           : ');
  readln(testmax);

  dodev(dev,zc,testmax);
  randomize;
  for count := 1 to 4 do      { preload array }
    tests[count] := random(2);

  { perform tests }

  for count := 1 to testmax do
  begin
{     if count mod 1000 = 0 then write('*');
 }  docount1(tests,one);
    docount2(tests,two);
    docount3(tests,three);
    docount4(tests,four);

    value := random(2);    { get next bit }
    shuffle(tests,value); { put next bit into test array }
  end;  { for count }
  writeln;

  write('Standard Deviation is ',zc:fore:aft);
  writeln('  Test maximum is ', testmax,' bits');
  getresult1(one,  testmax, dev[1], outfile);
  getresult2(two,  testmax, dev[2], outfile);
  getresult3(three,testmax, dev[3], outfile);
  getresult4(four, testmax, dev[4], outfile);

  close(outfile);

  continue;
end. { ttest }
```

Simulation of Maurer's Theoretical Method

```
program Maurer1(Input, Output);

{   8 bit bytes only
  this program is to take a file of data from a binary file and
  perform statistical tests on the data.
  performs tests on binary symmetric source using 8 bit non-overlapping
  pieces
}

uses
  crt;

const
     bits = 8;
     Values = 256;      { 2^bits }

type

  tstfile  = file of byte;
  filename = string[255];

var
  infile       : tstfile;            { source of binary string }
  outfile      : text;

  inpath,
  inname   : filename;            { choice of file names  }
  outpath  : filename;

  filelength,                      { length of file in bytes }
  initsteps,                       { length of initialising bytes }
  numsteps,                        { length of process bytes }
  count        : longint;         { counter }
  val          : byte;            { value from source }

  fTU,                            { avg log2 of numsteps terms }
  sum : real;                         { sum of  calculated values }

  Table   : array[0..255] of longint;    { time index of each char }

  finish  : boolean;
  ch : char;
{**********************************************************************}

procedure menu;
begin
clrscr;
 writeln('             TeDeum Text Statistical Program      ');
 writeln('              8 bit Maurer1 Randomness            ');
 Writeln; writeln;
end;             { menu }


{*******************************************************************}

begin                  { main }
  clrscr;
  menu;
  finish := false;
  while not(finish) do
  begin
    write('What is the Input file *.???  : ');
    readln(inname);

    inpath := 'c:\tedeum\maurtest\maurtst2\' + inname;
    assign(infile, inpath);
    reset(infile);

    filelength := filesize(infile);
    initsteps  := trunc(filelength/10);
    numsteps   := filelength - initsteps;

    writeln(inpath);writeln;
    outpath := 'c:\tedeum\maurtest\maurtst2\wttrump.res';
    assign(outfile, outpath);
    append(outfile);

    for count := 0 to 255 do      { clear the table }
      table[count] := 0;

    writeln('Reading Initial Values'); writeln;

    for count := 1 to initsteps do         { read initial values }
    begin
    if count mod 1000 = 0 then write('*');
      read(infile,val);
      table[val] := count;
    end;

    writeln('Performing Test ');writeln;

    for count := (initsteps + 1) to (initsteps + numsteps) do
    begin
      if count mod 1000 = 0 then write('*');
      read(infile,val);
      sum := sum + ln(count - table[val]);
      table[val] := count;
    end;
    writeln;writeln;

    fTU := (sum/numsteps)/ln(2.0);
    writeln('The Calculated Value For ',inpath,' is : ',fTU:3:7);
    writeln(outfile, inname,' ', fTU:3:7);
    sum := 0.0;
  close(infile);
  close(outfile);
  write('Continue ? spacebar');
  readln(ch);
  if ch <> chr(32) then finish := true;
  end;  { while not }
end.                        { main }
```

```
Simulation of Startpoints
program sim1;

{
  a program to generate a set of random sequences from the following
  1.        a set of startpoints
  2.        a set of selection sets of varying size
  3.        a set of randomisers

  the output sequences will be stored in a file for analysis

  note 1. all data will be short integers which will limit the
          increment steps to range 0-127

  author     david shaw
  date       12aug98
  version    1.0    initial design and implementation
}
{--------------------- Declarations --------------------------}

uses
      crt,      { use screen character routines }
      printer;  { use output to printer }

const errormessage1 = 'this option is not implemented';

      ranfilename   = 'd:\programs\simtest\';
      opfilename    = 'd:\programs\simtest\output\';
      spfilename    = 'd:\programs\simtest\spfile1.tst';
      ssfilename    = 'd:\programs\simtest\sset';
      outfilename   = 'd:\programs\simtest\out';

      randomisersize = 10000;      { size of randomiser file }

      noofrfiles         = 10;     { number of randomiser files }
      noofstartpoints    = 10;     { number of startpoints }
      noofselectionsets  = 10;     { number of selection sets }

      maxselectionsetsize = 128;   { largest size of selection set }
      maxoutputsetsize    = 128;

type
      outarray = array[1..maxoutputsetsize] of integer;
      outfile  = file of outarray;

      sintfile = file of shortint;  { a file of 0 & 1 for printing }
      spfile   = file of integer;   { a file of 0 to 255 }

      anyfilename = string[40];


var
      rfile     : sintfile;       { file of random bits }
      rfilename : anyfilename;    { name of random file in use }

{--------------------- functions ----------------------------}

function IntToStr(I: Longint): String;
{ Convert any integer type to a string }
var
 S: string[11];
```

```
begin
 Str(I, S);
 IntToStr := S;
end;

{--------------------- procedures ----------------------------}
procedure createrandomisers;
var count,
    count1 : integer;      { loop variable }
    val    : shortint;     { value to be stored in file }
    lfname : anyfilename;  { name of file  }
    lf     : sintfile;     { short integer file }
begin
  for count := 0 to noofrfiles-1 do
  begin
    { create file name }
    lfname := ranfilename + 'rmsr'+ inttostr(count) + '.tst';

    { create file }
    writeln('Creating : ',lfname);
    assign(lf,lfname);
    rewrite(lf);

    { fill with random data }
    writeln('Filling  : ',lfname);
    randomize;          { initialize the randomiser function }
    for count1 := 1 to randomisersize do
    begin
      val := random(2);
      write(lf,val);
    end; { for count1 }

    { close file }
    writeln('Closing  : ',lfname);
    close(lf);

  end; { for count }

end;  { create randomisers }

procedure createstartpoints;
var  count  : integer;     { loop variable }
     val    : integer;     { start point }
     lfname : anyfilename; { name of file }
     lf     : spfile;      { integer file }

begin
       { create file name }
    lfname := spfilename;

    { create file }
    writeln('Creating : ',lfname);
    assign(lf,lfname);
    rewrite(lf);

    { fill with random data }
    writeln('Filling  : ',lfname);
    randomize;          { initialize the randomiser function }
    for count := 1 to noofstartpoints do
    begin
      val := random(255);
      write(lf,val);
    end; { for count1 }

    { close file }
    writeln('Closing  : ',lfname);
    close(lf);

end; { createstartpoints }

procedure createselectionsets;
var  count,     { number of selection sets needed }
     count1,    { size of each selection set }
     setsize,   { number of integers in selection set }
     val        { set value }
             : integer;
    lfname   : anyfilename;
    lf       : spfile;

begin

  for count := 0 to noofselectionsets-1 do
  begin         { create file name }
    lfname := ssfilename + inttostr(count) + '.tst';

    { create file }
    writeln('Creating : ',lfname);
    assign(lf,lfname);
    rewrite(lf);

    { fill with random data }
    writeln('Filling  : ',lfname);
    randomize;          { initialize the randomiser function }

    setsize := random(255);  { generate size of selection set }
    for count1 := 1 to setsize do
    begin
      val := random(maxselectionsetsize);
      write(lf,val);
    end;  { for count1 }

    { close file }
    writeln('Closing  : ',lfname);
    close(lf);

  end; { for count }

end; { createselectionsets }

procedure createoutputfile;
var
   randomisers,
   startpoints,
   selections : integer;

   rfname,
   spname,
   ssname,
   opname  : anyfilename;

   rf      : sintfile;
   spf,ssf : spfile;
   opf      : outfile;
begin
  { select randomiser }
  for randomisers := 0 to noofrfiles-1 do
  begin
    rfname := ranfilename + 'rmsr'+ inttostr(randomisers) + '.tst';
    assign(rf,rfname);
    reset(rf);
    writeln(rfname);      { select randomiser file }


      { generate a sequence }
      ssname := ssfilename + inttostr(selections) + '.tst';
      assign(ssf,ssname);
      reset(ssf);
      writeln('    ',ssname);

      for selections := 0 to noofselectionsets-1 do
      begin    { select selection set }
        { generate outputfilename}
        opname := opfilename  + inttostr(randomisers) +
                  inttostr(selections)  + '.out';

        { create outputfile }
        writeln('      ',opname);
        assign(opf, opname);
        rewrite(opf);
        { generate outputsequences }


        { select bits using startpoint and selection set }
        { for each startpoint use selectionset to generate
          128 bits and store as a string in the output file }
        { get start point then with randomiser move to start
          point and select bit }
        { get selection set value and move to next bit and select bit }
        { ensure if index is greater than file size to return to
beginning. }


        { close outputfile }
        close(opf);
      end; { for selections }
      close(ssf);
    close(rf);
  end; { for randomisers }

end; { createoutputfile }

{---------------------- main program --------------------------}

begin { sim1 }
clrscr;
{ create files and open for writing }

{ create randomiser files  }
{  createrandomisers; }

{ create file of startpoints }
{  createstartpoints;}
{ create files of selection sets }
{ createselectionsets; }

{ create files of outputsequences from each randomiser and
  selection set with the startpoints provided }
createoutputfile;

{ close files }

end. { sim1 }
```

```
Ordering Paradigm
program order;
{
  a program to test the ordering paradigm on random sequences.

}
uses crt;

const  noofbits = 50;


type
   arrayline = (message,key,encryption);

   anyarray = array[arrayline,1..noofbits] of integer;

   intfile = file of integer;

   filename = string[40];

var
  test : anyarray;

  infilename : filename;
  infile     : intfile;

  knum0,               { number of 0's in key }
  knum1,               { number of 1's in key }
  enum0,               { number of 0's in encryption }
  enum1,               { number of 1's in encryption }
  MEmatch,             { match between message bit and encrypted bit }
  bitstested : longint;

  kp0,
  kp1,
  ep0,
  ep1,
  psame      : real;
{--------------------------------------------------------}

procedure displayarray(var a : anyarray);
var count : integer;
begin
  writeln;
  for count := 1 to noofbits do
    write(a[message,count]);
  writeln;
  for count := 1 to noofbits do
    write(a[key,count]);
  writeln;
  for count := 1 to noofbits do
    write(a[encryption,count]);
  writeln;
end;

procedure getmessage(var a : anyarray );

var count : integer;
begin
  randomize;
  for count := 1 to noofbits do
```

```
    begin

      a[message,count] := random(2);

      a[key,count]   := 0;
      a[encryption, count] := 0;
    end; { for }
  end; { getmessage }

  procedure getencryption(var inf : intfile;
                          var   a : anyarray;
                          var n0,
                              n1,
                              e0,
                              e1,
                              bt  : longint);
  var i, count : integer;
  begin
    if not eof(inf) then
    begin
      gotoxy(75,3); write('²');
      for count := 1 to noofbits  do
      begin
        read(inf,a[key,count]);
        inc(bt);
           { increment key counts }
        if a[key,count] = 1 then inc(n1) else inc(n0);
           { perform encryption }
        if  a[message,count] = a[key,count] then a[encryption,count] := 0
        else a[encryption,count] := 1;
        if a[encryption,count] = 1 then inc(e1) else inc(e0);

      end;
       gotoxy(75,3); write('Û');

    end; { while }

  end; { getencryption }


  procedure testarray(var a   : anyarray;
                      var mem : longint);
  var count, num : integer;

  Begin
    num := 0;

    for count := 1 to noofbits do
    begin
      if a[message,count] = a[encryption,count] then inc(num);
    end; { for }

    mem := mem + num;
  end; { testarray }

  {--------------------------------------------------------}

  begin
    textcolor(yellow);
    textbackground(blue);
    clrscr;
  writeln(' ':25,'Ordering paradigm program ');
  writeln;
  write('What is the name of the input file : ');
  readln(infilename);
  writeln;
  gotoxy(74,2);write('ÉÍ»');
  gotoxy(74,3);write('º º');
  gotoxy(74,4);write('ÈÍ¼');

  assign(infile,infilename);
  reset(infile);
  { generate a test sequence }
  getmessage(test);

  Mematch := 0; bitstested := 0;
  knum0 := 0;       knum1 := 0;
  enum0 := 0;       enum1 := 0;

  while not eof(infile)do
  begin
    getencryption(infile,test, knum0,knum1, enum0,enum1, bitstested);
    testarray(test,MEmatch);
  end; { while }

  psame := MEmatch/bitstested ;

  kp0 := knum0/bitstested;
  kp1 := knum1/bitstested;
  ep0 := enum0/bitstested;
  ep1 := enum1/bitstested;

  writeln;
  writeln;
  writeln('The number of bits tested   : ',bitstested);
  writeln;
  writeln('    Key Stream ');
  writeln('Key P(0)  = ',kP0:4:6);
  writeln('Key P(1)  = ',kp1:4:6);
  writeln;
  writeln('    Encryption');
  writeln('Enc P(0)  = ',eP0:4:6);
  writeln('Enc P(1)  = ',ep1:4:6);
  writeln;


  writeln('The number of matches between Message and encryption :
',MEmatch);
  writeln;
  writeln('Proportion of matches between Message and Encryption :
',psame:4:6 );

  close(infile);
  write('press enter to continue');
  readln;
end.  { order }
```

9.17. Enumeration of all Samples

			Random Number							
percent	sum	select	0	1	0	1	0	0	1	1
50.00%	2	4	-	-	-	-	0	0	1	1
75.00%	3	4	-	-	-	1	-	0	1	1
75.00%	3	4	-	-	-	1	0	-	1	1
50.00%	2	4	-	-	-	1	0	0	-	1
50.00%	2	4	-	-	-	1	0	0	1	-
50.00%	2	4	-	-	0	-	-	0	1	1
50.00%	2	4	-	-	0	-	0	-	1	1
25.00%	1	4	-	-	0	-	0	0	-	1
25.00%	1	4	-	-	0	-	0	0	1	-
75.00%	3	4	-	-	0	1	-	-	1	1
50.00%	2	4	-	-	0	1	-	0	-	1
50.00%	2	4	-	-	0	1	-	0	1	-
50.00%	2	4	-	-	0	1	0	-	-	1
50.00%	2	4	-	-	0	1	0	-	1	-
25.00%	1	4	-	-	0	1	0	0	-	-
75.00%	3	4	-	1	-	-	-	0	1	1
75.00%	3	4	-	1	-	-	0	-	1	1
50.00%	2	4	-	1	-	-	0	0	-	1
50.00%	2	4	-	1	-	-	0	0	1	-
100.00%	4	4	-	1	-	1	-	-	1	1
75.00%	3	4	-	1	-	1	-	0	-	1
75.00%	3	4	-	1	-	1	-	0	1	-
75.00%	3	4	-	1	-	1	0	-	-	1
75.00%	3	4	-	1	-	1	0	-	1	-
50.00%	2	4	-	1	-	1	0	0	-	-
75.00%	3	4	-	1	0	-	-	-	1	1
50.00%	2	4	-	1	0	-	-	0	-	1
50.00%	2	4	-	1	0	-	-	0	1	-
50.00%	2	4	-	1	0	-	0	-	-	1
50.00%	2	4	-	1	0	-	0	-	1	-
25.00%	1	4	-	1	0	-	0	0	-	-
75.00%	3	4	-	1	0	1	-	-	-	1
75.00%	3	4	-	1	0	1	-	-	1	-
50.00%	2	4	-	1	0	1	-	0	-	-
50.00%	2	4	-	1	0	1	0	-	-	-
50.00%	2	4	0	-	-	-	-	0	1	1
50.00%	2	4	0	-	-	-	0	-	1	1
25.00%	1	4	0	-	-	-	0	0	-	1
25.00%	1	4	0	-	-	-	0	0	1	-
75.00%	3	4	0	-	-	1	-	-	1	1
50.00%	2	4	0	-	-	1	-	0	-	1
50.00%	2	4	0	-	-	1	-	0	1	-
50.00%	2	4	0	-	-	1	0	-	-	1
50.00%	2	4	0	-	-	1	0	-	1	-
25.00%	1	4	0	-	-	1	0	0	-	-
50.00%	2	4	0	-	0	-	-	-	1	1
25.00%	1	4	0	-	0	-	-	0	-	1
25.00%	1	4	0	-	0	-	-	0	1	-
25.00%	1	4	0	-	0	-	0	-	-	1
25.00%	1	4	0	-	0	-	0	-	1	-
0.00%	0	4	0	-	0	-	0	0	-	-
50.00%	2	4	0	-	0	1	-	-	-	1
50.00%	2	4	0	-	0	1	-	-	1	-

	25.00%	1	4	0	-	0	1	-	0	-	-
	25.00%	1	4	0	-	0	1	0	-	-	-
	75.00%	3	4	0	1	-	-	-	-	1	1
	50.00%	2	4	0	1	-	-	-	0	-	1
	50.00%	2	4	0	1	-	-	-	0	1	-
	50.00%	2	4	0	1	-	-	0	-	-	1
	50.00%	2	4	0	1	-	-	0	-	1	-
	25.00%	1	4	0	1	-	-	0	0	-	-
	75.00%	3	4	0	1	-	1	-	-	-	1
	75.00%	3	4	0	1	-	1	-	-	1	-
	50.00%	2	4	0	1	-	1	-	0	-	-
	50.00%	2	4	0	1	-	1	0	-	-	-
	50.00%	2	4	0	1	0	-	-	-	-	1
	50.00%	2	4	0	1	0	-	-	-	1	-
	25.00%	1	4	0	1	0	-	-	0	-	-
average	25.00%	1	4	0	1	0	-	0	-	-	-
50.00%	50.00%	2	4	0	1	0	1	-	-	-	-
total = 70											

The list above indicates all the possible combinations of four bits taken from the random eight bit sequence. The average of each sequence is calculated and the average value of the average is equal to the average of the random sequence or population.

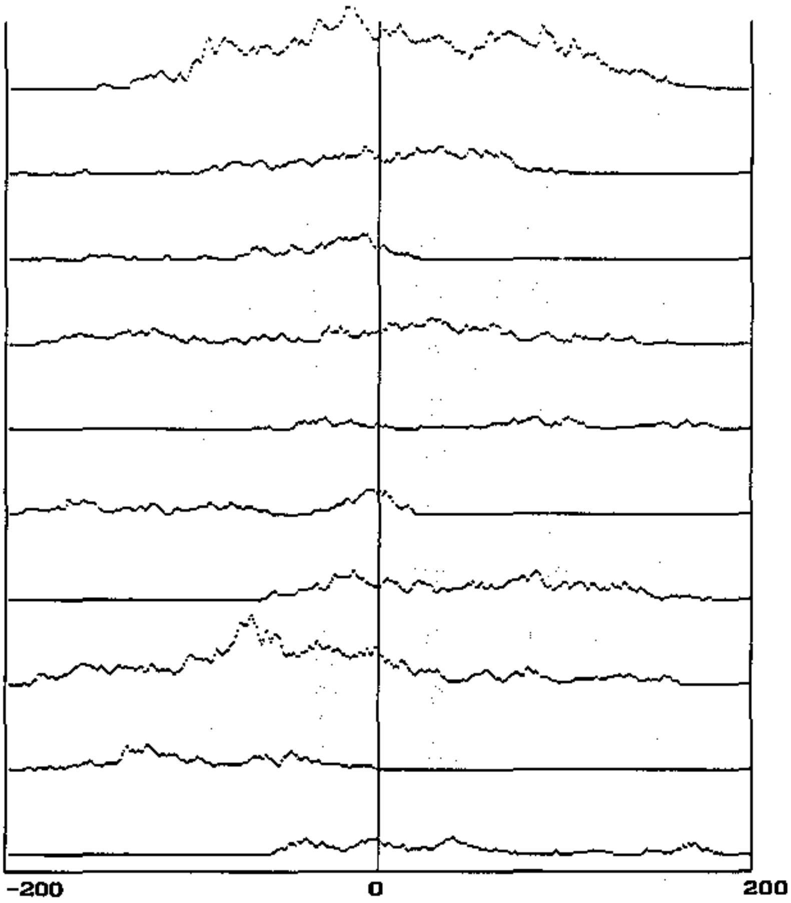


Figure 9-1 Traces of Ten Single Bit Runs

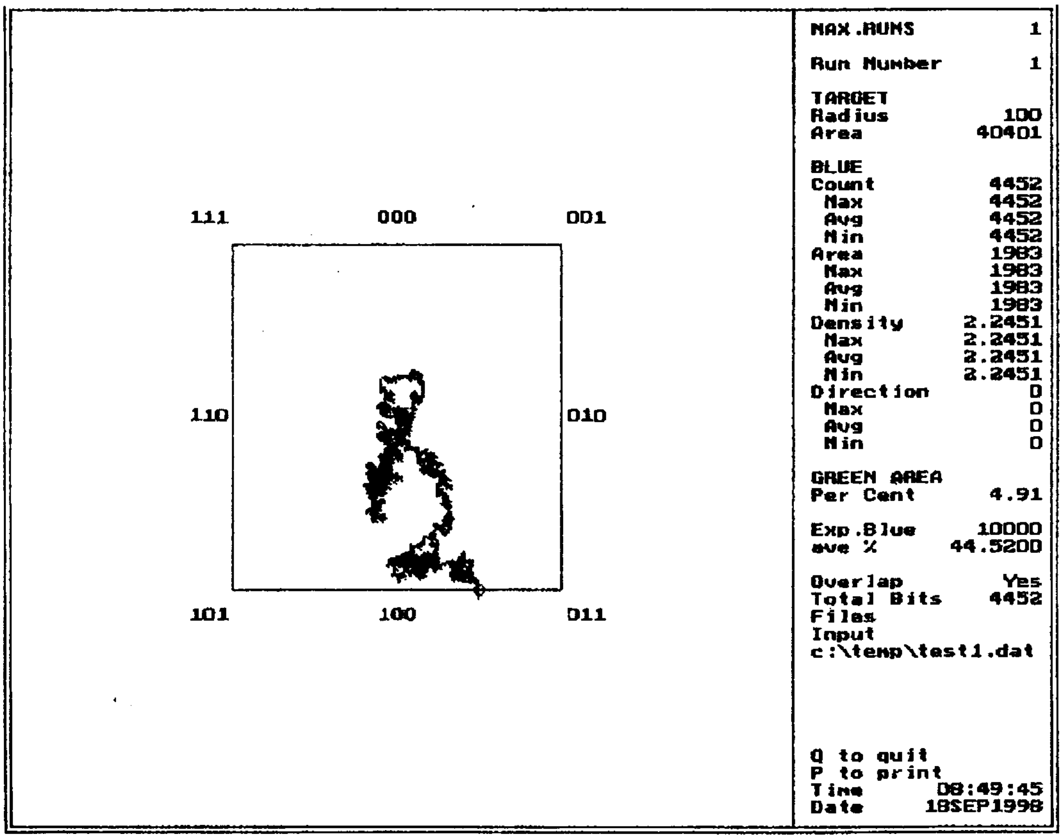
9.18. Data Visualisation Results

In figure 9.1 , each row is 401 elements long, that is, 200 elements either side of a central element. The trace for 10 consecutive runs are plotted. The vertical dimension is scaled and each trace is offset vertically to display all the traces with minimum interference between them.

It is noted that six of the traces continue to a termination and favour one side almost to the exclusion of the other. Three of these traces terminate on the negaive side and three on the positive side.

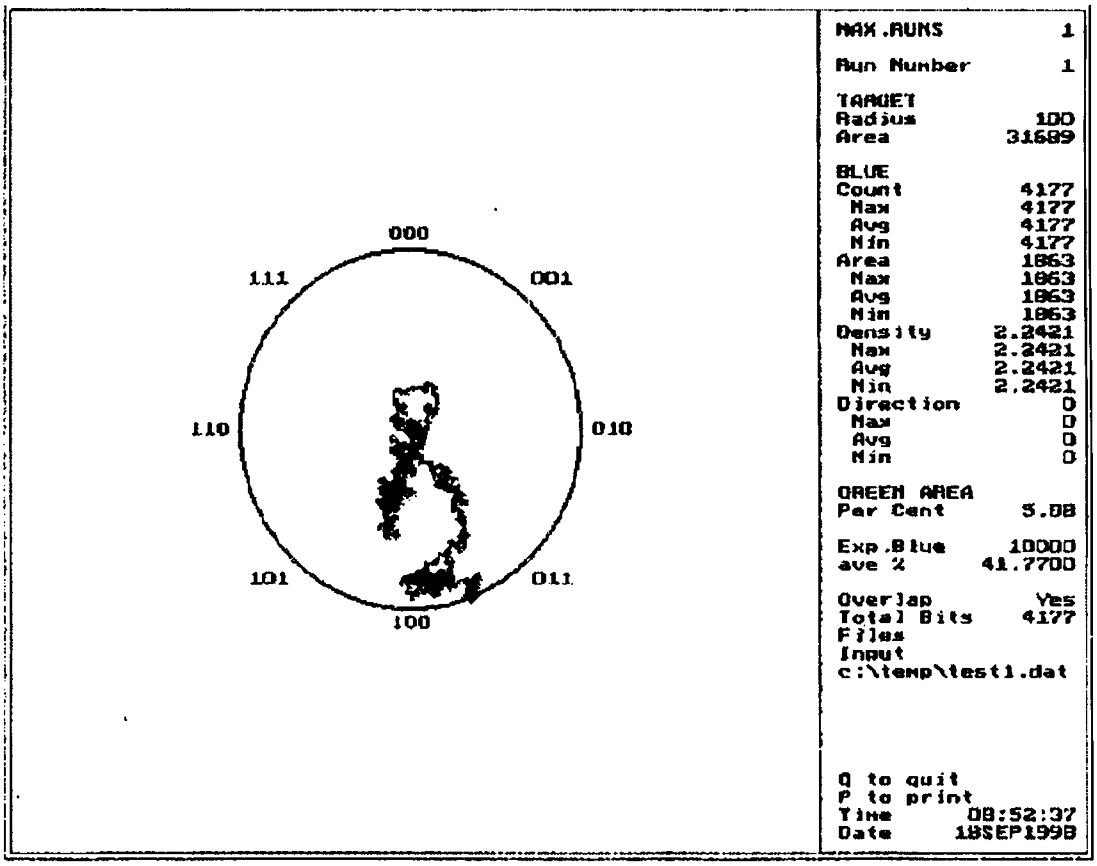
This trace is the instantaneous average of the previous ten traces scaled to fit. It can be seen that the area under the trace tends to be larger to the left of the central axis indicating a preference for 0's.

The location and number of peaks may be used to characterise the output of a generator.



```
MAX.RUNS              1

Run Number            1

TARGET
Radius              100
Area              40401

BLUE
Count              4452
 Max               4452
 Avg               4452
 Min               4452
Area               1983
 Max               1983
 Avg               1983
 Min               1983
Density          2.2451
 Max             2.2451
 Avg             2.2451
 Min             2.2451
Direction             0
 Max                  0
 Avg                  0
 Min                  0

GREEN AREA
Per Cent           4.91

Exp.Blue          10000
ave %           44.5200

Overlap             Yes
Total Bits         4452
Files
Input
c:\temp\test1.dat




Q to quit
P to print
Time           08:49:45
Date          18SEP1998
```

Figure 9-3 Square 3-bit Pattern



```
MAX.RUNS              1

Run Number            1

TARGET
Radius              100
Area              31689

BLUE
Count              4177
  Max              4177
  Avg              4177
  Min              4177
Area               1863
  Max              1863
  Avg              1863
  Min              1863
Density          2.2421
  Max            2.2421
  Avg            2.2421
  Min            2.2421
Direction             0
  Max                 0
  Avg                 0
  Min                 0

GREEN AREA
Per Cent           5.08

Exp.Blue          10000
ave %           41.7700

Overlap             Yes
Total Bits         4177
Files
Input
c:\temp\test1.dat



Q to quit
P to print
Time           08:52:37
Date          18SEP1998
```

Figure 9-4 Round 3-bit Pattern

9.19. Determining a Basic Sequence

The implementation of the test relies on progressively choosing overlapping sequences of a particular length and counting the number of 'ones' in the sequence. To identify the length of a rotated sequence means being able to determine the number of bits per sequence is the same.

For example, using a sequence ℵ' and the number of blocks, k = 2 * n
note: matching numbers that are n values apart are a possible basic sequence size.

ℵ' = 111-11101000-00011101-00111010-01110100-11010001-101

n = 6, k = 12
111111, 111110, 111101, 111010, 110100, 101000,
010000, 100000, 000000, 000001, 000011, 000111

bit count per sub-sequence
6,5,5,4,3,2,
1,1,0,1,2,3 no matches at interval of 6

n = 7
1111110, 1111101, 1111010, 1110100, 1101000, 1010000, 0100000,
1000000, 0000001, 0000011, 0000111, 0001110, 0011101, 0111010

6,6,5,4,3,2,1,
1,1,2,3,3,4,4 3 matches at interval 7, generate each 7 bit block and count the bits until there is a difference then stop.

1111110 1000000 *1110100 1110100 1110100 1101000 1101* = 6,1
1111101 0000001 *1101001 1101001 1101001 1010001 101* = 6,1
1111010 0000011 *1010011 1010011 1010011 0100011 01* = 5,2
1110100 0000111 *0100111 0100111 0100110 1000110 1* = 4,3
1101000 0001110 1001110 *1001110 1001101 0001101* = 3,3,4
1010000 0011101 *0011101 0011101 0011010 001101* = 2,4
0100000 0111010 *0111010 0111010 0110100 01101* = 1,4

n = 8

11111101, 11111010, 11110100, 11101000, 11010000, 10100000, 01000000, 10000001,
00000011, 00000111, 00001110, 00011101, 00111010, 01110100, 11101001, 11010011

7,6,5,4,3,2,1,2,
2,3,3,4,4,4,5,5 4 matches at interval 8 then try each 8 bit sequence.

11111101 00000011 *10100111 01001110 10011010 001101* = 7,2,
11111010 00000111 *01001110 10011101 00110100 01101* = 6,3,
11110100 00001110 *10011101 00111010 01101000 1101* = 5,3,
11101000 00011101 00111010 01110100 11010001 *101* = 4,4,4,4,4
11010000 00111010 *01110100 11101001 10100011 01* = 3,4,
10100000 01110100 *11101001 11010011 01000110 1* = 2,4,
01000000 11101001 *11010011 10100110 10001101* = 1,5,
10000001 11010011 *10100111 01001101 0001101* = 2,5,

The Basic Sequence length is apparently 8 as the value 4 is repeated for each block in the sequence. It is also shown that the partial block size at the beginning is three bits.

n = 9

111111010, 111110100, 111101000, 111010000, 110100000, 101000000, 010000001, 100000011, 000000111,
000001110, 000011101, 000111010, 001110100, 011101001, 111010011, 110100111, 101001110, 111111010

7,6,5,4,3,2,2,3,3,
3,4,4,4,5,6,6,5,7 4 matches at interval 9 then try each 9 bit sequence

111111010 000001110 *100111010 011101001 101000110 1* = 7,3
111110100 000011101 *001110100 111010011 010001101* = 6,4
111101000 000111010 *011101001 110100110 10001101* = 5,4
111010000 001110100 111010011 *101001101 0001101* = 4,4,5
110100000 011101001 *110100111 010011010 001101* = 3,5
101000000 111010011 *101001110 100110100 01101* = 2,6
010000001 110100111 *010011101 001101000 1101* = 2,6
100000011 101001110 *100111010 011010001 101* = 3,5
000000111 010011101 *001110100 110100011 01* = 3,5

n = 10

1111110100, 1111101000, 1111010000, 1110100000, 1101000000, 1010000001, 0100000011, 1000000111, 0000001110, 0000011101, 0000111010, 0001110100, 0011101001, 0111010011, 1110100111, 1101001110, 1010011101, 0100111010, 1001110100, 0011101001

7,6,5,4,3,3,3,4,3,4,
4,4,5,6,7,6,6,5,5,5,   5 matches at interval 10 then try each 10 bit sequence

1111110100 0000111010 *0111010011 1010011010 001101* = 7,4,
1111101000 0001110100 *1110100111 0100110100 01101* = 6,4
1111010000 0011101001 1101001110 *1001101000 1101* = 5,5,6
1110100000 0111010011 *1010011101 0011010001 101* = 4,6
1101000000 1110100111 *0100111010 0110100011 01* = 3,7
1010000001 1101001110 *1001110100 11010001101* = 3,6
0100000011 1010011101 *0011101001 1010001101* = 3,6
1000000111 0100111010 *0111010011 010001101* = 4,5
0000001110 1001110100 *1110100110 10001101* = 3,5
0000011101 0011101001 *1101001101 0001101* = 4,5

9.20. External and Ancillary Systems

Resource: Equipment ID and Automatic Mileage Reporting
Product: AT5770 Odometer/Identification Tag
Supplier: Amtech Systems Corp .

Amtech now offers fleets an automatic identification tag with the added capability to *9.21*.company notes include the ability to prognostically react to the maintenance needs of a fleet, as well as to accurately monitor and control fuel costs.

Contact: Amtech, (972) 733-6600, Fax: (972) 733-6699.

Resource: Total Connectivity Solution
Product: Cadec 4000Plus
Supplier: Cadec Systems

The Cadec 4000Plus information system combines several newly released products from Cadec into one "value package" designed to improve virtually every aspect of information management within a fleet operation. The system can be configured to fit a broad range of specific customer needs.

At the heart of the 4000Plus package is Cadec's DataBridge technology. DataBridge, the company explains, is a communications gateway designed to cost-effectively link the engine, its subsystems, the driver and vehicle through an onboard computer to the company's home office and its remote locations. Products such as RouteMessenger and RemoteOffice can be used with the DataBridge technology. A number of methods for transferring data also are possible, and cellular and PCMCIA technologies, as well as ATA J1587/1708 datalink and sensor processing capabilities, are also featured.

Another key element of the Cadec 4000 Plus is OfficeEnabler, which offers a range of advanced information management and decision support tools. In addition to core analysis capabilities for performance, productivity and regulatory compliance, OfficeEnabler features RouteOnBoard, VehicleVision electronic tachograph and Cadec's new StepAhead program series.

New sensor and datalink capabilities of the system, Cadec reports, also add dimension to the "connectivity gateway." The 4000Plus's sensor capability, for example, allows for on/off status determination and monitors specific information, such as refrigeration, power take-off and dual speed on the rear axles. The Cadec 4000Plus also reads and monitors the datalink for critical information regarding engine performance, including the odometer and tachometer. Fleet managers can use the information about vehicle and driver performance to maximize the overall efficiency of their operation.

Contact: Cadec Systems, (800) 25Cadec in the U.S., (800) Cadec20 in Canada, Fax: (603) 623-0604,
URL: http://www.cadecsystems.com:8080
E-mail: Info@cadecsystems.com

Resource: Refrigeration Performance Monitoring
Product: Data Management System
Supplier: Thermo King Corporation

Fleets are assisted in monitoring and controlling cargo temperature and trailer humidity conditions with Thermo King's Data Management System (DMS). The microprocessor-based information gathering and storage device can be added to practically any refrigerated trailer or truck for more efficient and profitable temperature management, the company says. Another benefit is reduced maintenance and operating costs, since a review of the data from DMS can point out inefficiencies or trends that can be corrected before major problems occur.

DMS provides fleets managers with a range of information by monitoring, recording and displaying temperature readings for up to six sensors, including return and discharge air. It also monitors and records six operating functions, including heat, cool, defrost, high speed, low speed and Cycle Sentry, and it is equipped to monitor and record up to four additional conditions, such as door openings. The result, Thermo King states, is a complete record of the trip, from start to finish.

Desired operating conditions can be programmed in DMS via a keypad on the device or with a personal computer. An alarm system alerts the driver should an out-of-range condition develop during the trip. DMS accumulates data at pre-selected intervals, which can range from every two minutes to every four hours. Since data is maintained in a non-volatile memory that requires no power supply, stored information cannot be erased or lost.

An internal battery with a life of up to ten years, the company adds, keeps the internal clock working even if the refrigeration unit is shut down. A plug-in port allows data retrieval whenever desired. Data can be output to a hand-held printer or personal computer.
Contact: Thermo King Corp., (612) 926-3754, Fax: (612) 926-0943.

Resource:	Yard Management
Product:	Yard*Man I
Supplier:	Message Xpress

Installing a radio frequency identification (RFID) system to monitor fleet assets in yards or terminals is simplified with Yard*Man I, a fully integrated hardware, software and monthly fleet activity reporting service. The hardware consists of an RFID tag reader and tags which can be detected by a single reader at distances up to 1,400 ft. Tags are attached to a truck or trailer, and readers are placed in yards to be monitored. The PC-based software runs on a Windows 3.X operating system. Fleet managers can access historical and real-time fleet activity instantly via PC and a modem. Any number of locations can be monitored at the same time using just one PC.

According to the company, Yard*Man I saves fleets time by automatically performing yard inventories. In addition to simply showing whether a particular asset is in one of the monitored facilities, the software enables daily yard activity reports to be produced so fleet managers can track just-in-time performance, reconcile driver logs, schedule and track maintenance operations and track leased asset utilization. The reporting service included with Yard*Man I provides two, complete yard management reports per day on all asset activity. Real-time yard inventory inquiries are available at an additional cost.

Contact: Message Xpress, (800) 637-7248, Fax: (800) 270-2823.

Resource: Vehicle Tracking
Product: MicroTracker
Supplier: American Technologies

The MicroTracker remote vehicle-locating system combines a Global Positioning System receiver, radio transceiver and RF modem all in one compact unit. According to the company, it is an inexpensive, reliable and accurate tracking module. Vehicles, trailers and intermodal containers in which the device is installed, ATI advises, can be instantly located anywhere in North America. When combined with the company's LOGITRAK mobile satellite communications software, MicroTracker offers real-time communications and vehicle tracking from a local Internet connection to the dispatcher's computer.
Contact: American Technologies, (414) 922-7030, Fax: (414) 922-7011,
E-mail: amtech@landnet.com. .

Resource: Rear Vision System
Product: Surround Sight
Supplier: Clarion

Rear vision cameras and in-cab monitors designed to match a fleet's individual needs are available from Clarion. The camera lineup features a waterproofing system with an internal O-Ring seal to protect the connection. A special rain gutter and recessed lens, the company states, prevents water from interfering with the transmission of a crystal-clear picture. Included on the CC-830G50 model is an automatic motorized lens cover and built-in heater which prevent the camera from fogging up and keep it operational in foul weather. All cameras feature a die cast aluminum housing, which makes them resistant to shock and vibration, and a 1/3-inch CCD imager that provides clear views even in low lighting conditions.

Clarion monitors feature a shock-resistant casing and a glare-resistant picture tube that provides 400+ lines of resolution for greater clarity. A night dimmer feature enables easy night time viewing. Every system, the company notes, is capable of multi-zone monitoring. Additional camera switchers can be used with any monitor to hook up multiple cameras. The CJ-770F model, however, includes a built-in two-camera switcher for use with rear and right view cameras. This model also features electronic on-screen AcuSight markers to help drivers judge backing distances more accurately. All Clarion monitors can be adapted for RightSight usage, which automatically turns cameras on when a driver shifts into reverse or activates the right turn signal.
Contact: Clarion, (310) 327-9100, Fax: (310) 327-1999.

Resource: Accident Prevention
Product: Blind-Sight
Supplier: Collision Avoidance Systems, Inc

Comprised of intelligent sensors that are installed in various locations, the Blind-Sight collision avoidance system can be equated to electronic eyes watching the front, back and sides of a vehicle. Mounted in the truck cab is the command module, a microprocessor-based unit that computes all the sensor information and alerts the driver audibly and visually when objects are detected in a blind spot.

When a vehicle is first started, Blind-Sight performs a self-check to verify that all sensors and on-board computer systems are working properly. At the same time, it checks to see if any objects are in the driver's blind spot in front of the truck. On the road, right-side sensors are activated whenever the driver engages the right turn signal. The Blind-Sight backup system is activated when a driver shifts into reverse. On the dashboard, the command module features an imbedded outline of a truck with indicator lights that correspond to sensors located on the side of the tractor and trailer, as well as on the rear of the trailer. When an object is sensed, the location of the object is displayed as well as the distance in feet and inches. Optional driver's side coverage is also available and is activated by engaging the vehicle's four-way flasher.

Also offered with Blind-Sight is an Intelligent Mirror System that consists of driver's and passenger side mirrors that are mounted on standard mirror brackets. Information relating to obstacles in the blind spots to the rear and sides of the truck is displayed in these mirrors with large numbers and warning lights.

Contact: Collision Avoidance Systems, (800) 780-9062, Fax: (770) 475-8322.

Resource:	Accident Prevention
Product:	Echovision
Supplier:	Armatron International

To eliminate the risk for accidents associated with lane changes, merging and backing maneuvers, the company offers an obstacle detection system featuring automatic activation and intuitive audible and visual warnings. Echovision's rear system alerts the driver with a low pitched interrupted tone when an obstacle is detected at about 10 ft from the truck. The tone increases in frequency as the vehicle gets closer to the object, changing to a high-pitched continuous tone at approximately 4 ft. Sensors mounted to the right side of the truck monitor the blind area on this side of the vehicle, sending a signal to the small box with warning light that is mounted on the right side of the cab where it can be seen when the driver looks at his right-hand mirror. If the right turn signal is on while an object is in the driver's blind spot, Echovision will also sound an alarm.

To minimize distractions to the driver, Echovision features a Smart Program that enables it to distinguish between true blind side obstacles and false warnings, such as from road-side objects, guard rails, Jersey barriers, trees and signs. The system also incorporates a continuous self-test for the electronics and actual transmission and reception of the sensors to ensure that the entire system is operational. If any part of the system malfunctions, the system fault light will warn the driver. A short series of tones occur as the system is turned on to verify the audio warning components are functioning properly.

Contact: Armatron International, (617) 321-2300; Fax: (617) 321-2309.

Resource:	Trailer Identification
Product:	TailTag
Supplier:	Navigato International

With TailTag fleets can obtain the same information about the location and status of trailers as they do for tractors. Details about the coupling or uncoupling of a trailer are also provided. The information can be relayed to a fleet's home base or registered in the tractor. The product does not use radio frequency communication. TailTag is easily installed on the tractor, using existing electrical wiring and requiring no special skills or tools. The system supports all relevant proposed and existing alpha-numerical formats for trailer identification.

Contact: Navigato International, Denmark, +45 9751 3788, Fax: +45 9751 4050
E-mail:NATOFVM@po.ia.dk
Web page: http://www.datashopper.dk/~finth/navigato.html.

Resource:	Vehicle Tracking
Product:	MIRAS 1100
Supplier:	SPS Technologies

The MIRAS 1100 vehicle remote control and tracking device uses 220MHz two-way radio technology for voice capability as well as data transmission. This technology, the company explains, replaces the cellular dependency of the original MIRAS (Mobile Interactive Remote Activated Solutions) unit, saving the end user 40% or more on monthly operating costs. The unit features fully interactive tracking and remote control capabilities to allow fleets to track vehicles and take remote control of many vehicle operating variables such as turning a vehicle on and off. MIRAS uses GPS technology to track vehicles within 10 meters and transmit the data back to a fleet's in-house PC.

Zoom in and out features provided with the street level, detailed map allow users to track vehicles along their route on a computer screen. The system also enables fleets to monitor a vehicle's engine functions, speed, cargo temperature and security systems on the office PC, and it helps drivers in need of directions find their next destination. Interactive capabilities include being able to change a cargo's thermostat from the office or instantly re-route a vehicle.

Contact:SPS Technologies, (800) 320-1186, Fax: (954) 677-9132.

Resource:	Accident Prevention
Product:	Safety Alert
Supplier:	Cobra Electronics Corporation

Installed on emergency response, construction or public utility vehicles, Safety Alert is designed to improve highway safety by automatically transmitting warning signals to the radar detectors of motorists within a ¾-mile radius of the emergency or potentially hazardous road situation. The Cobra Safety Alert transmitter, the company explains, sends out several distinctive signals depending on the situation— one set of signals for a stationary road hazard versus another set for a moving emergency vehicle, for example. The safety signals, Cobra notes, can be picked up by all radar detectors currently on the market.

In addition, "intelligent" receivers that display warning messages on special LCD screens or by using LED patterns also are available from Cobra. Safety Alert is FCC approved. It is currently in use in 30 U.S. states.

Contact: Geltzer & Company, (212) 575-1976, Fax: (212) 245-2145.

Resource:	On-Board Electronic Scale
Product:	Dynaload
Supplier:	Dynacraft

The Dynaload on-board scale is designed to help fleets maximize loads for greater profitability, while eliminating lost revenues from overweight fines and commercial scale fees. According to the company, Dynaload can determine gross, payload and axle group weights on all tractor-trailers equipped with air suspensions. For axle groups not equipped with air suspensions, the Dynaload system can be supplied with load-cell sensors.

On air-equipped vehicles, the system uses pneumatic sensors to detect minute changes in air pressure, Dynacraft notes, and is accurate to within 1%. A sensor connected to the air suspension leveling valve relays analog data to a solid-state transmitter mounted on the frame rail. The transmitter converts the signal to digital and relays the data directly to Dynaload's microprocessor-based meter in the truck's cab. Just one sensor/transmitter per axle is required, according to the company. Installation takes between three and four hours. Dynaload can be ordered on new Kenworth and Peterbilt tractors. It is also available through Kenworth and Peterbilt dealers for installation in the aftermarket on any truck make.

Contact: Dynacraft, (800) 532-6550, Fax: (206) 351-3041; or contact a nearby Kenworth or Peterbilt dealer.

Resource: Obstacle Detection
Product: SCAN System
Supplier: Electronic Controls Company (ECCO)

SCAN is an ultrasonic sensory device designed to assist drivers in safely changing lanes, backing up or docking. Comprising the system are three primary components— the smart sensor, the driver alert module and the main power harness. According to the company, SCAN's smart sensors feature state-of-the-art transducers which detect vehicles or objects near the truck's perimeter. The software interprets the signal received and transmits the data along a single wire to a display module mounted inside the cab. The module visibly and audibly alerts drivers to obstacles within SCAN's detection zone.

Ultrasonic transducers, ECCO states, provide broad detection-area coverage and are capable of withstanding harsh environmental elements. For trouble-free operation, the company adds, the smart sensors perform self-diagnostics and report any malfunctions. They are also designed to monitor ambient temperatures and will activate a heating element to prevent buildup of ice and snow during winter use.

Contact: Electronic Controls Company, (800) 635-5900, Fax: (208) 376-3410.

Resource: Ergonomic Dashboard
Product: Driver's Cabin
Supplier: Argo Instruments

In cooperation with VDO Kienzle— developer of the Integrated Driver Information System in use on motor coaches in Europe— Argo Instruments introduces a driver's cabin designed for improved efficiency and safety of transit bus and motor coach operations in North America. The ergonomic cockpit, the company says, includes a computerized, integrated driver instrument panel which serves as the central information hub. With the exception of the speedometer and prescribed warning lights, all other vital display functions are included on one easily readable LCD monitor. Pre-programmed controls allow information to be prioritized and presented according to its importance to the operator.

The driver's cabin dashboard also is designed to adjust with the steering wheel, making it easily adaptable to the needs of different drivers. In addition to improved visibility and less information clutter in the cockpit, the company notes, another benefit of the dashboard's design is less physical stress on drivers, including reduced incidence of back strain. Additional features and options that are being designed for the future, Argo reports, include real-time links to fleet headquarters and trip monitoring capabilities.

Contact: Argo Instruments, (540) 665-0200, Fax: (540) 662-2127.

Resource: Data Collection
Product: XATA Distribution Information System
Supplier: XATA Corporation

Four basic components comprise the XATA Distribution Information System that, according to the company, can help improve fleet productivity. The Driver Computer is the vehicle on-board system, with touch-screen computer, which interacts with the driver to electronically capture and communicate vehicle operating data. The Data Station is located at trip origins and destinations, such as warehouses, terminals and distribution centers. It is used to download dispatch data to and collect trip information from the Driver Keys.

Each electronic Driver Key carries a driver's individual ID number and log records. It is used to exchange dispatch and trip data between the Driver Computer and Data Station. The fourth component, the Fleet Management System (FMS), is PC software designed to help fleet managers evaluate and report trip details. The FMS collects information from Data Sta-tions and sends route dispatch information to the Driver Computer via the Data Stations.

Contact: XATA Corp., (612) 894-3680, Fax: (612) 894-2463.

Resource: Dispatching
Product: PEN*KEY 6000 Series
Supplier: Norand Corporation

Using mobile computing technology, Norand's PEN*KEY 6000 Series of products are designed to help fleets improve delivery planning and information management, as well as increase dock and driver productivity. Fleets can choose between the 22-ounce 6100 computer or the full-screen 6600 unit, depending on the extent of functions and price of system they desire. The complete dispatch system includes P&D route planning, graphics maps with driver instructions, strip and load manifests, customer calls for pickup, linehaul planning, dock planning, real-time delivery reporting, signature capture and driver production management.

An accurate cross-docking system, Norand adds, eliminates misloads and improves efficiency through an optional bar-code scanner, on-line re-weigh program, automatic trailer door closing, visual load diagram, and other features. Wireless communications to printers via an optional IrDA (infrared link) are also possible. Both units are small and easily carried from the truck to docks or delivery sites, the company notes, and can also be worn on a belt for even greater portability.

Contact: Norand Corp., (800) 4-PENKEY, Fax: (319) 369-3453.

Resource: Satellite Communications
Product: Enhanced Display Unit for OmniTRACS
Supplier: Qualcomm

An enhanced display unit for the OmniTRACS satellite mobile communications system is available from Qualcomm. Used to send and receive text messages, the unit is mounted in the truck's cab and features a user-friendly keyboard and display screen. The enhanced display unit is compatible with the OmniTRACS Mobile Communications Terminal (MCT) as well as a new OmniTRACS on-board recor-der, which is planned for release later this year.

A smart card reader/writer feature, the company notes, will allow drivers to store and retrieve data on small plastic cards when used with the on-board recorder. The display operates over a wide temperature range and is vibration and shock protected. It features a 15-line by 40-character screen that permits both text and high definition graphics.

Contact: Qualcomm, Inc., (619) 658-2000, Fax: (619) 658-1578.

Resource: Collision Warning
Product: Forewarn Side Detection System
Supplier: Delco Electronics Corp

The Forewarn Side Detection System (SDS) is an advanced safety system for Class 6 through 8 vehicles designed to warn drivers of vehicles moving in the blind area along the right side of their tractor and trailer. SDS, the company explains, uses microwave radar technology to detect moving objects and target-discrimination software to minimize false alarms from stationary objects sensed in its path.

The sensor, mounted on the tractor to the rear of the right-hand door, monitors approximately an 8-ft zone. Whenever a moving object is detected in the zone, Delco advises, the unit sets off a visual alert via a flashing light located on the outside side-mirrors of the vehicle. An audible warning, triggered by a lane change signal, alerts drivers to moving objects i n the blind spot zone. According to the manufacturer, testing of the Forewarn system has proven it reliable even in harsh weather conditions.
Contact: Delco Electronics Corp., (317) 451-5700, Fax: (317) 451-5426.

Resource: Data Transmission
Product: Radio Frequency Identification System
Supplier: Amtech Systems Corp.

Using Amtech's radio frequency identification (RFID) technologv, fleets can automate many operations easily and securely, the company says. RFID facilitates collecting, tracking and reporting data for maintenance and other fleet management areas. Hands-free and paperless data communications applications for the automatic vehicle and equipment identification system include gate control access, yard inventory, fuel terminal authorization, automatic weigh-in-motion scales, preventive maintenance monitoring and intermodal equipment tracking.

Contact: Amtech, (214) 733-6600, Fax: (214) 733-6699.

Resource: Interactive Fleet Management
Product: RoadTrac
Supplier: RoadTrac, Ltd

According to the company, RoadTrac integrates Global Positioning Satellite tracking technology, proprietary mapping software and state-of-the-art digital cellular communications to provide fleets with accurate vehicle locating combined with a truly interactive voice, data and vehicle security management tool. RoadTrac features a display screen for easy viewing of all critical data, including vehicle latitude, longitude and heading; vehicle speed; street address; vehicle ID, license number, make and model; and alarm status and security mode. In-vehicle security sensors and enabling device options, the company says, are designed to provide instant response to "real-time" conditions.

Programmable command functions include: Lock/Unlock Cargo Doors, Enable Flashers/Disable Engine, Audio or Video Monitoring, Voice Recognition, Operator Panic/Distress Mode, and PC and Fax Connectivity. Other features include on-screen routing, bi-directional text messaging, interactive Key Cards, computer aided dispatch, mapping overlays of vehicles and custom report generation, including state fuel tax reporting.

Contact: RoadTrac Ltd., (800) 661-4971, Fax: (770) 446-3897.

Resource: On-Board Recorder with GPS
Product: DataTrax/GPS
Supplier: Rockwell Transportation Electronics

The Rockwell DataTrax/GPS combines functions of the Tripmaster on-board recorder with a built-in Global Positioning System (GPS) receiver. DataTrax/GPS, the manufacturer reports, can act as a stand-alone recorder of vehicle data or be used with a variety of input and display devices, including alphanumeric wired and portable keypads, pen-based computers and touch-screen units. The system uses GPS to add the date, time and vehicle position to traditional OBC data. According to the company, a key feature is the ability to accomplish hands-free recording of state line crossings and accurate mileage tracking by jurisdiction for fuel tax reporting. Fleets can also incorporate DataTrax/GPS data with popular route analysis and planning software.

Contact: Rockwell Transportation Electronics, (800) 997-2595, Fax: (515) 247-2812.