Edith Cowan University

Research Online

Theses: Doctorates and Masters

Theses

1-1-1997

Randomised shuffle and applied misinformation: An enhanced model for contact-based smart-card serial data transfer

Michael Collins
Edith Cowan University

Follow this and additional works at: https://ro.ecu.edu.au/theses

 Part of the Computer Sciences Commons

Recommended Citation

Collins, M. (1997). *Randomised shuffle and applied misinformation: An enhanced model for contact-based smart-card serial data transfer*. https://ro.ecu.edu.au/theses/877

This Thesis is posted at Research Online.
https://ro.ecu.edu.au/theses/877

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.

- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).

- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

USE OF THESIS

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

"Randomised Shuffle and Applied Misinformation: An Enhanced Model for Contact-Based Smart-Card Serial Data Transfer"

by

Michael Collins B. App. Sc. (Hons).

A dissertation to be submitted in fulfilment of the requirements for the degree of

Master of Science (Computer Science) by Research.

Department of Computer Science,
School of Mathematics, Information Technology and Engineering,
Edith Cowan University,
Perth, Western Australia.

Supervisors:
Dr T. J. O'Neill and Professor A. C. Watson.

Monday, 3 November 1997

Abstract

Contact-based smart-cards, which comply to the International Standard ISO-7816, communicate with their associated read/write machines via a single bi-directional serial link. This link is easy to monitor with inexpensive equipment and resources, enabling captured data to be removed for later examination. In many contact-based smart-cards the logical abilities are provided by eight-bit microcontroller units (MCU) which are slow at performing effective cryptographic functions. Consequently, for expediency, much data may be transferred in plain-text across the vulnerable communications link, further easing an eavesdropper's task.

Practitioners in military communications protect transmitted information by varying a link's carrier frequency in an apparently random sequence that is shared secretly between the sender and the authorised receiver. These multiplexing techniques, known as frequency or channel-hopping, serve to increase the task complexity for and/or confuse potential eavesdroppers.

The study seeks to ascertain the applicability and value of protection provided by channel-hopping techniques, when realised with minimal additional overhead of microcontroller resources to the contact-based smart-card communications link. The apparent randomised shuffling of data transferred by these techniques has the potential benefit of deterring those observers who may lack the equipment and expertise to capture and decode the communicated message.

Table of Contents

Declaration

I declare that this thesis does not incorporate without acknowledgement any material previously submitted for a degree 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 acknowledgement is made.

Signature:

Date: 3d November 1997

Acknowledgements

No significant project can be completed by one person. Accordingly, I would like to thank all those people who helped and encouraged me with this project. However, there are some special people to whom I wish to express particular gratitude.

Thank you Sue, Ben and Rebecca, for always being there when I was here.

Thank you Dr Thomas O'Neill, for your guidance, and because you can inspire the transformation of a mediocre passage of text into something that is worth writing.

Thank you Professor Tony Watson, for your enthusiasm, your inspiration, and for never losing sight of the overall picture.

Thank you Henry Hickling, for the diligence with which you proof-read this dissertation. As always, you are a true friend in time of need.

Thank you the team at RD2P, for the encouragement and suggestions early in the study, and for the hospitality you extended to this stranger.

Acknowledgement and thanks must go to Phil Tang and Andrew McDonald, for the serial communications assembly code adapted for use in the study.

1. Introduction

This chapter contains an introduction to smart-cards and describes their origins and evolution; discusses their relationship with security; and identifies that member of the smart-card family whose interface is of particular concern. The aims of the study are itemised, the research question is stated, and a synopsis of the strategy to be pursued is given.

1.1 Background to the Study

A smart-card may suitably be defined as "*A card of ISO dimensions which has in-built logical ability*" (Lathom-Sharp, 1995). This definition is useful in that it acknowledges the two main features of smart-cards: namely, that standards are in place to govern the form and interface of the cards; and that each card contains an ability to process data, some or all of which must be communicated to/from the card. The study is concerned with the privacy of communication of that data, the reasons for which will be developed in the remainder of this chapter.

A smart-card may be seen as a token which possesses on-board processing power, and may carry information about its holder. It has evolved from electronically readable tokens that emerged in 1960, when airline tickets at Chicago's O'Hare airport had magnetically encoded data attached to them (Hutcheon, 1992). Hutcheon further reports that the use of magnetic data swipe cards had grown by 1991 to an estimated US circulation in excess of one billion, while Twentyman (1997) reports that MasterCard alone has almost 370 types of credit/debit cards in circulation. Hutcheon further suggests that the growth was primarily due to the widespread acceptance of Automated Teller Machines (ATM) and Electronic Funds Transfer at Point of Sale (EFTPOS) facilities. This widespread acceptance has led to the popular adoption of the standard $3^3/_8$" by $2^1/_8$" plastic card favoured by the largest providers, the banks and credit companies.

Initially, the magnetic strip was used to help combat fraud whereby data (e.g., a credit limit) might be written onto the special tape moulded in the reverse of the card. Today, magnetic swipe cards are in widespread use to control access in a physical or logical sense, and in the main, they are used to distinguish between a person who is authorised for access and one who is not. Baker (1991, p. 127) describes three basic ways to make this distinction:-

(i) using something the person has (*e.g., the card itself*);

(ii) ascertaining what the person knows (*e.g., a PIN*); and

(iii) establishing whom the person is (*some unique identifier e.g., a fingerprint*).

When accompanied by the additional information of a Personal Identification Number (PIN), the use of swipe cards can satisfy only items (i) and (ii) above. The conventional magnetic swipe card, however, is limited in the amount of data that it can reliably contain, preventing it from carrying sufficient data to perform item (iii). Furthermore, data contained in magnetic stripe cards may be easily read, edited and copied, using inexpensive equipment, thereby limiting the cards' ability to enact secure transactions. In particular, such limitations affect off-line operations, this being one of four requirements identified by Ferguson (1994, p. 318) for an effective electronic cash system. Specifically, these requirements are:-

" - **Security**. Every party in the electronic cash system should be protected from a collusion of all other parties (multi-party security).

- **Off-line**. There should be no need for communications with a central authority during payment.

- **Fake privacy**. The bank and all other shops should together not be able to derive any knowledge from their protocol transcripts about where a user spends her money.

- **Privacy** (untraceable). The bank should not be able to determine whether two payments were made by the same payer, even if all shops co-operate."

Criminal elements find it easy to take advantage of the limitations of magnetic stripe cards, leading to a level of fraud which Longley (1994, p. 497) estimates to cost society hundreds of millions of dollars. The perceived scale of this criminal activity is illustrated by Fox (1993), who recounts "The banking card system is teetering on the brink of collapse. The banks know it and they are just hanging on for as long as they can, hoping that they can get a new system up and running before the fallibility of the old system is exposed." Assuming successful trials, it appears likely that the new banking card system will be based upon smart-cards as they offer superiority, in terms of security and functionality (Longley, 1994, p. 497) over their passive magnetic predecessors.

There have been successful long-term, high-volume smart-card trials in France, with some twenty-two million cards issued (Lathom-Sharp, 1995). Ongoing trials exist in other regions such as Australasia, where Visa announced the availability of Stored Value Cards (SVC) in Australia and New Zealand commencing July 1995 (ComputerWorld, 31 March 1995), and where MasterCard will run a pilot scheme for sixteen kilobyte memory SVCs in Canberra, A.C.T. from October 1995 (Financial Review, 3 April 1995).

Industry shares the view that smart-cards will succeed magnetic stripe cards in banking: Hovenga, the smart-card technical marketer for Motorola's Microcontroller Division in Austin, Texas, USA, quoted by Hodgson (1995), says "Probably the application that is going to drive smart-cards here is financial/banking." Indeed, smart-cards will be used in many diverse systems which require positive user identification. *What's New in Telecommunications* (March/April, 1995, p. 15) describes the dependency of the European Digital Network upon smart-cards to provide subscriber information; *What's New in Electronics* (April, 1995, p. 55) outlines their incorporation into complete network security models for Australian military applications; and Johnson & Tolly (1995) detail tests performed on several commercial smart card products used in Token Authentication applications for corporate networks.

The concept of an intelligent (smart) card was conceived by Jurgen Dethloff in 1969 and first patented in Japan in 1970 by Prof. Arimura (Townend, 1995). In 1974, the French inventor Roland Moreno patented worldwide the concept of a plastic card containing microelectronic processing power (Longley 1994, p. 498), to provide the card's intelligence. Today, smart-cards are endowed with an embedded microprocessor and memory sufficient for their particular purpose. By example, the smart-card's memory may be used to hold the unique identification data necessary to establish Baker's (1991, p. 127) "who the person is" attribute, which may then be evaluated against output from on-site sensors, e.g., fingerprint or retina scanners.

At first, smart-card microprocessors and memory devices comprised separate standard components, and even today's versions will typically contain an eight-bit Central Processing Unit (CPU) of 1980s origin, e.g., those of the 8051, 68HC05, or H8 CPUs (Townend, 1995). Recognising, however, the peculiar security needs of smart-cards and the size of the potential market, major electronic organisations have tailored the microelectronics to the task in hand. Paterson (1991, p. 29) says that security, in relation to smart-cards, "can be grouped into three main categories:

- designed in (*intrinsic*) security;
- manufacturing security; and
- application security."

Briefly, the *intrinsic security* relates to the situation of the components used. If there exist control, data and address wires between components of the on-board microelectronics, then there is the possibility of attack by:

- invasion of the plastic outer sheathing;
- monitoring of their electromagnetic emissions; or
- scanning via electron microscopes (Ferreira, 1990, p. 338).

It is considered better, therefore, to place all microelectronic components on the same silicon die, which, by minimisation of inter-component wiring, may then be made less vulnerable to such attacks.

Manufacturing security relates to the controls that are placed over the production and testing of the smart-cards, so that they may be neither altered nor researched during this phase of their lives. Typically involved will be the physical securing of production facilities and data, and the provision of internal one-time fusible links to remove permanently the microelectronics from any test modes when production is complete (Paterson, 1991, p. 33). Such test modes are used to determine the quality of the silicon chips at several stages of their manufacture and necessarily provide access to every detail of the chip under test.

Application security falls into the domain of the Software Engineer, who may take advantage of any hardware provided features to secure the card provider's application software and/or data. According to Vedder (1992), the prevalent eight-bit processors in smart-cards limit their cryptographic abilities to those of challenge and response methods employing DES-like shared-key algorithms: "Public-Key techniques are not yet feasible for smart-cards." More recently, however, it is evident that this limitation may be combatted by incorporation of additional circuitry or firmware, notes Dinnissen (1995). The extra circuitry may relieve the CPU of specialised software burdens, such as an intelligent building Local Area Network (LAN) interface (Hodgson, 1995) or encryption processes. Circuitry for the latter may take the form of a "modular multiplication module" (Morita, 1990, p. 406), or similar mechanism to assist in the processing of relatively long-key algorithms such as that created by Rivest, Shamir and Adleman (RSA). During 1994, Phillips B.V. announced (Phillips, 1994) their DX smart-card which employs an "optimised calculation unit" to perform five hundred and twelve bit RSA computations directed toward secure digital signature generation. Despite the extra processing power, each signature computation takes approximately five hundred milliseconds (Phillips, 1994).

It is appropriate to describe the concept of security in terms of context and perspective. Landwehr, Heitmeyer & Mclean (1984) provide the following definition: "A system is *secure* if it adequately protects information that it processes against unauthorised disclosure, unauthorised modification, and unauthorised withholding... We say *adequately* because no practical system can achieve these goals without qualification; security is inherently relative." Historically, in the case of magnetic stripe credit-cards, it is the author's experience that the card providers exercise total control over the data that is secured. In the commercial area, as observed by Roberts (1991), "the integrity of information is often of paramount importance, whereas the military field requires confidentiality as its first priority." Now recall that current smart-cards incorporate relatively slow eight-bit CPUs (Townend, 1995) which, in turn, determine that cryptographic operations on communications with them will be slow. Vedder (1992) suggests that processing a sixty-four bit data block using DES will typically require fifteen milliseconds with a five megahertz CPU clock, and that those sixty-four data bits will take ten milliseconds to transmit at an (asynchronous) baud rate of ninety-six hundred bits per second. Given also that communication bandwidth is an expensive resource, the author considers it likely that the card providers will secure adequately only that information which is precious to them - unless legislation mandates otherwise for protection of the cardholders. Furthermore, Fitzsimmons (1995), a contributor in the field of Computers and the Law, states, "The laws of Australia do not recognise any general right of privacy, whether by way of common law or by way of constitutional rights"; and, "In addition, there are many misconceptions concerning privacy. ... People do not have a right to control information about themselves." In Australia, at least, the situation is unlikely to change. Hilvert (1997) quotes the statement of John Howard, Australian Prime Minister, on 21st March 1997 "the Commonwealth will not be implementing privacy legislation for the private sector" and suggests that "the message for the private sector is that if it is not specifically legislated against, it is OK to continue to develop and exploit personally intrusive databases, data mining and privacy probes with little federal oversight". Fitzsimmons further indicates that the situation in other countries and states with regard to personal privacy, may currently be as unprotective as Australia's, a view shared by De Schutter (1991) who states "the individual becomes the weakest link in the new information technology era, not only from a sociological and economic

point of view..., but equally from the angle of legal protection (privacy)." It is, therefore, conceivable that data, which a cardholder may reasonably desire to remain secure, is communicated as plain-text in order to maintain a specified performance from a card whose computational power is provided by an eight-bit processor that is slow to perform cryptographic functions. This lack of privacy is noted by Dinnissen (1995) who states "cards reveal the card identity and data content to any reader or anyone tapping communications."

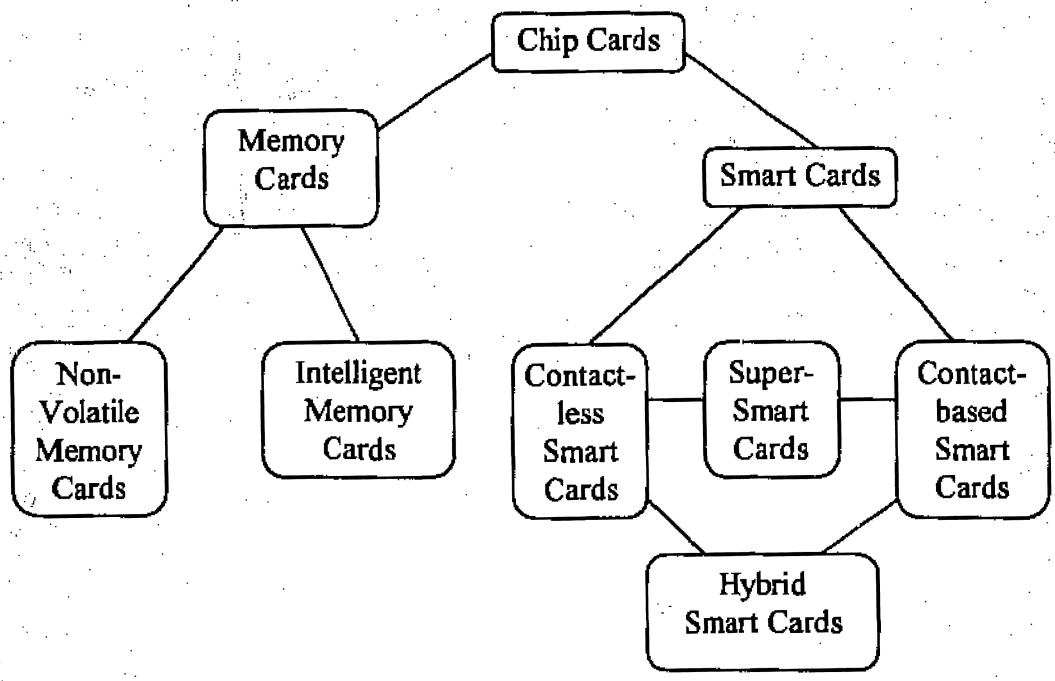

Figure 1.1. The Smart Card Family and their near relatives - Memory cards.

Smart-cards may be distinguished from memory cards by their ability to perform logical operations in addition to storing data. Figure 1.1 indicates that smart-cards assume one of two main forms: contactless or contact-based. Offshoots from these include super smart-cards which incorporate keys and a liquid crystal display (LCD) in order that the cardholder may interact directly with the card, and hybrid cards which are magnetic swipe cards additionally equipped to facilitate either contactless and contact-based smart-card operation. Contactless cards communicate via a modulated electro-magnetic field; hence, they may be accessed from distances varying from several centimetres (for battery-less cards) to fifty metres along a line of sight (for high frequency, self-powered, cards) (TIRIS, 1994, p. 7). Contact-based cards may only derive power and communicate via a set of pads, which are exposed on one face of the card, when mating with a purpose built connector.

The applications of contactless cards are many and diverse: e.g., they may be found tracking products during production/warehousing, or the monitoring of time and performance data for transport operators (TIRIS, 1994, p. 7). Contactless cards are currently under consideration by the U.S. Army for wartime logistics management on a global basis, destined within five years to be read by satellite (Van Order, 1995). The chief attractions of contactless cards, as described by Hook (1995), are that they provide reliable non-contact operation; are impervious to most dirt and contaminants; are environmentally durable; may be recoded in-situ via coded radio signals; and, can exhibit very low initial costs (from US$0.40c each) and maintenance costs. The widespread acceptance of such cards, however, is seen by Hook (1995) as being limited by the following impediments: lack of recognised *de facto* or international standards with respect to operating frequencies or communications protocols; and, a susceptibility to electro-magnetic interference which is not yet governed by widespread or uniform standards. These impediments have led to a market that is populated by incompatible systems which often require special antenna or circuitry in order to achieve satisfactory performance in particular environments or geographical areas of operation. Although there are many documented instances of contactless smart-cards being used for small financial transactions such as ticketing and incentive schemes (Seidman, 1995), Cordonnier (1995) sees the fact that contactless cards may be accessed remotely as a factor against their being readily accepted by consumers for larger financial transactions.

Contact-based smart-cards can be accessed only when inserted into, and connected electrically to, a suitable reader. Three major card issuers are Visa, Mastercard and Europay (VME). Since 1993, these have sought to establish standards that will allow merchants and users to manage smart-cards (Hoffman, 1995). Given that the banking industry currently provides a substantial user base for the smart-card's predecessor, the magnetic stripe card, then it is likely, as indicated in Hodgson (1995), that the emergent bank driven VME standard will propel smart-cards into widespread use. Now that major card providers have aligned themselves with the VME standard (Dancer, 1995) the credit-card proportioned, contact-based, ISO-7816 conformant smart-cards may proliferate internationally

Although smart-cards may contain different application software and operating systems, the ISO-7816 standard for smart-cards mandates that card processing machinery will present a standard mechanical and electrical interface to the cards, and it homogenises the protocol via which communications are effected. ISO-9992 provides for standardisation of the "messages between integrated circuit card and the card accepting device" (Vedder, 1992). If standardised smart-cards replace magnetic stripe cards as instruments of identification for access control, then many agencies may access, observe, and potentially alter their contained data. In order to protect against misuse of that data, it then becomes necessary to apply security measures such as encryption, the effectiveness of which, as found by Monod (1995), may vary between no security to a very high level of security.

When probing the security of an entity or system, it is appropriate to identify points of weakness. One apparent weak point in the highly standardised ISO-7816 smart-cards is the single contact used as the communications interface between the card and its associated read/write processing machine. Using simple and readily available circuitry, as in Figure 1.2, it is possible to eavesdrop, unobtrusively and without detection, upon the entire conversation between the card and the read/write machine. Having collected the data from the conversation, the eavesdropper may attempt decryption upon it at leisure. Conversely, contactless smart-cards, because of their diverse non-standardised designs and complex reader circuitry, render widespread monitoring of their communications difficult and/or unattractive.

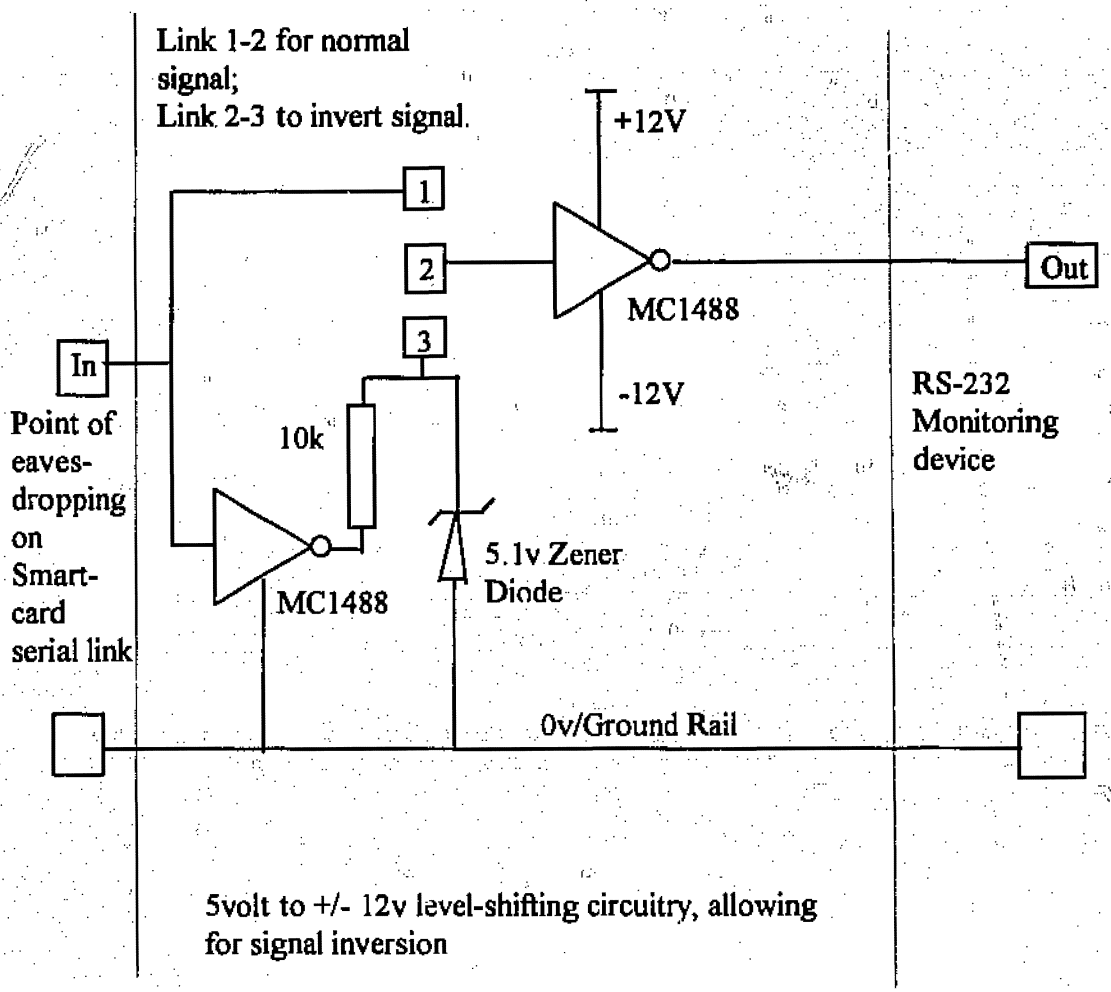

Figure 1.2. **Example of circuitry necessary to monitor contact-based smart-card to read/write machine communications interface.**

The smart-card communications interface may be thought of as being a single channel along which all data must pass for the duration of a conversation. The fields of military communications and radar are subject to similar problems when restricting their transmissions to a single channel. The confidentiality, integrity or availability of their communications channel may be compromised when monitored, eavesdropped or blocked. One approach to lessen such a compromise, in addition to encryption, is to use a mechanism known as frequency-hopping, where the data is conveyed upon carriers whose frequency is altered according to a pattern known only to the legally communicating parties. The interloper's task is further complicated by the concurrent transmission of invalid data on those carriers not being used to convey valid data or by using time division multiplexing (TDM) of the data on a carrier - i.e., patterned interspersing periods of valid data and invalid data. Each carrier frequency or pattern of time division may be seen as providing an independent communications channel via which data may be switched, and a direct analogy may be drawn between frequency-hopping or time-division hopping (or a combination of the two) and channel-hopping.

Returning our thoughts to the contact-oriented smart card, the author suggests that a channel-hopping mechanism, or randomised shuffle of transmitted data, might be implemented without placing excessive demands on the eight-bit CPUs upon which the cards currently depend. Such a mechanism would rely upon the valid data being multiplexed across several channels according to a one-off pattern shared only between card and read/write machine, and would increase the complexity of an eavesdropper's task in terms of required equipment and/or methods of message detection. The result of this increased eavesdropper's task complexity would mean that additional protection might be provided economically for transmitted data which otherwise may have been seen as expendable in the face of the high CPU resource costs associated with effective encryption.

1.2 Significance of the Study

From the background discussion, the following is apparent. On the one hand, the ISO-7816 compliant contact-based smart-cards rely upon an easily monitored communications interface between card and read/write machine using half-duplex asynchronous character transmission techniques. Transmissions using this interface may be captured for later inspection by using inexpensive and easily obtained circuitry, an example of which is illustrated in Figure 1.2. Privacy is not guaranteed, therefore, for the communications link between card and read/write machine, and encryption currently provides the sole means of achieving security for the card-holder's communicated data. On the other hand, military communications engineers, in addition to encrypting their data, have evolved systems of frequency-hopping to complicate significantly the task of those who wish to interfere with or monitor their communications channels. In the above context, each frequency to which the communications link hops equates to a separate channel along which communicated data flows for a duration agreed by the communicating parties.

In the main, as found by Townend (1995), ISO-7816 compliant cards employ eight-bit microcontroller architectures to provide the on-card logical functionality. Such logical functionality, together with the ability to hold significant amounts of data, as suggested in Hodgson (1995), may lead to their adoption as the successors to the magnetic swipe-cards which are in widespread use today to control access. However, as Vedder (1992) suggests, their eight-bit microcontrollers, whilst providing economical and proven technology, are comparatively slow in performing the calculations required for effective encryption.

Importantly, the card-holder has little or no control over what elements of his/her data are held or transmitted in a secure manner; and, as Fitzsimmons (1995) implies, the law does not offer card-holders consistent levels of protection. Depending upon the implementation, Monod (1995) found that inherent smart-card system security levels may range from none to very secure.

Now consider that magnetic swipe cards may be used for applications unforeseen by their original supplier. For example, building upon an employee's reluctance to part with their personal credit card, it is currently possible to use information contained on credit cards to implement building access control (Kirkpatrick, 1995). If smart-cards become used in similar circumstances, it is then possible that the card will be inserted into many different read/write machines, each one of which presents an opportunity for communicated data to be monitored and subsequently inspected using simple circuitry.

This study aims to demonstrate the possibility of adapting channel-hopping techniques to complicate the task of an eavesdropper who wishes to record the conversations between a contact-based smart-card and its read/write machine. Successful outcomes of the study will afford the smart-card users, holders and manufacturers a number of potential advantages:-

- a more secure channel for *all* communicated data between card and read/write machine;
- little or no additional circuitry;
- low software overhead;
- minimal CPU overhead; and
- existing contact-based smart-card CPU designs may remain compliant with ISO-7816.

1.3 Purpose of the Study

The study investigates the possibility of using mechanisms whereby channel-hopping may be implemented economically, in terms of hardware and software, by ISO-7816 compliant smart-cards, and to evaluate the mechanisms' effectiveness at confusing a potential communications link eavesdropper. Various techniques will be explored, and assessed, for randomly shuffling/hopping the valid data across several channels. The study will aim to demonstrate an improvement of security offered by such "randomised shuffle" mechanisms, and their cost in terms of hardware, software and CPU resources, when used to shield data transmitted via the contact-based smart-card's communications link from eavesdropping.

1.4 Research Question

The main question:-

"Can the mechanism of channel-hopping be adapted to enhance the security of data transmitted between an ISO-7816 compliant contact-based smart-card and its read/write machine with minimal cost to CPU resources, hardware and software?"

The major components of the above question are :-

a) "What elements of channel-hopping can be utilised in the context of an ISO-7816 compliant smart-card?"

b) "What channels can be exercised to provide hopping routes?"

c) "Can the elements found in answer to *a)* be used in conjunction with the channels found in *b)*?"

1.5 Summary and synopsis of the remainder of the study

Security, as defined by Landwehr et al. (1984), is a relative, qualified concept. Qualification of security may differ when viewed from the perspective of card-issuer or card-holder. The discussion in this chapter seeks to indicate that a vulnerable point exists in the communications link between a contact-based smart-card, which may be poised to succeed the credit card as an access token, and an appropriate read/write machine with which the card is obliged to communicate. Due to this vulnerable point, smart-card issuers may choose, for expediency, to communicate data in unencrypted form. It is noted that military communications practitioners, when faced with similar vulnerability, have adopted a practice of channel-hopping via patterned alteration of the carrier frequency. This serves to increase the complexity of monitoring or interference and so enhances the security of their communications links. The study sets out to answer the question of whether the channel-hopping techniques may be successfully applied, whilst consuming minimal CPU resources, to contact-based smart-cards.

Chapter two takes the form of a review of the relevant literature. Previous work, illustrated in the firm foundation of text books and augmented by the documented research and experiences given in papers and articles, forms the basis of guidance and justification for the approach taken by, and substance of, the proposed study. It is demonstrated that the underlying presence of a weak communications link may enable more fundamental threats, such as information leaking. The nature of frequency/channel-hopping is explored, together with its applications, and a justification is established for its adaptation to provide a solution to the problem outlined in the above chapter. Attacks typical of those which the solution is anticipated to endure are identified, based upon knowledge of attacks rendered upon conventional methods of coding and ciphering.

Chapter three describes the research design. The fundamental design goals are outlined, together with an evaluation of potential improvements recognised at this stage. There follows the descriptions of the specific procedures to be developed in the pilot software implementation. Examples are provided in the form of scenarios, accompanied by appropriate pseudo-code developed for these.

Chapter four presents the findings and results of the pilot implementation. Selection criteria for a suitable target system are discussed and an overview of the features offered by the eventual target is given. The test programs of sender and receiver, which demonstrate treatment of the design goals, are examined. Suitable extracts from the test programs are presented, together with their productions of randomised shuffle, and are used to provide answers to the research questions.

Chapter five concludes the study. A summary is given of its beginnings and initial aims; the manner in which the project framework was arrived at; the design criteria for the pilot implementation; and, the resultant product of the randomised shuffle implementation. Finally, implications are discussed for current practice and future research of the method's potential uses in the large.

This document concludes with appendix A, consisting of a comprehensive glossary of terms used; appendix B, where implementation source code appears; and a section where end text references used to support the study are given.

2. Review of Relevant Literature

2.1 General Literature

Torrieri (1992, p. 291) relates that the interception of communications data may be attempted, in general, for many diverse reasons such as: reconnaissance, surveillance, position fixing, identification, or as a prelude to jamming. Of these, the author suggests that those of reconnaissance, identification and surveillance may apply to parties interested in intercepting a smart-card's transmitted data, implying concern for passive attacks rather than active ones as the physical nature of the smart-card's contact to receiving socket mechanism renders undetected on-line insertion of modified data difficult. Torrieri's described interception system includes three basic functions which must always be achieved: *detection, frequency estimation* and *direction finding*. Relating these to the context of a contact-based smart-card's half-duplex communications link reveals that:

- *detection* may be achieved by observing activity on the link, (i.e., when the signal moves from its steady state);
- *frequency estimation* may also be determined by observation. ISO-7816-3 (ISO/IEC 7816-3, 1994, Clause 6.1.1) mandates that cards with an internal clock shall use a bit rate of 9600 bits per second (b.p.s.), whilst externally clocked cards are prescribed an initial bit period (in seconds) of $(^{372}/_{external\ clock\text{-}frequency})$; and
- *direction finding* (of the transmitting party) may be discerned by recognition of "lulls" in link activity (announcing that a change of direction is imminent) and by examination of the conversation delimiters and/or headers which are mandated in ISO standards (ISO/IEC 7816-3 Part 3, 1994).

As Fumy (1991) states, "It is generally regarded a simple matter to record the data passing through a communications line without detection by the communicating parties." Eavesdropping, or the interception and recording of data for examination, may enter two of De Schutter's (1991) five categories, shown in Figure 2.1, into which incidence of computer abuse may be placed: namely, data espionage and unauthorised interception.

Category	Example/description
Manipulation of data	e.g., input of false data, alteration, erasure, deterioration and/or suppression of stored data or programs with fraudulent intent.
Data espionage	unlawful collection or acquisition and/or use of data.
Computer sabotage	leading to the destruction or disruption of software or hardware.
Unauthorised access or interception	of a computer and/or telecommunications system with infringement of security measures.
Program piracy	e.g., the infringement of the exclusive right of the owner of a protected computer program.

Figure 2.1. De Schutter's (1991) categories of computer abuse.

More formally, the interception of communicated data represents an underlying threat to system security which Ford (1994) has included in his hierarchy of threats. Ford's hierarchy indicates that underlying threats may enable more fundamental ones. Three stages of identification or classification are necessary for assembly of the hierarchy: namely,

- *fundamental threats*, those typically associated with the confidentiality, access and integrity of data or resources;
- *primary enabling threats*, the realisation of which may lead to an enabling of the more fundamental threats; and
- *underlying threats*, the presence of which may enable more fundamental threats.

Figure 2.2, adapted from Ford's explanations, provides illustration by example of each classification of threat.

Classification	Type	Example/description
Fundamental threats	Information leakage →	Information is disclosed to unauthorised entities.
	Integrity violation →	Consistency of data is compromised through unauthorised editing
	Denial of Service →	Legitimate access to resource is deliberately impeded
	Illegitimate use →	Unauthorised use of resource
Primary enabling threats	Masquerade →	An unauthorised entity poses as an authorised entity, so obtaining rights and privileges of the authorised entity
	Bypassing controls →	An attacker exploits security flaws or weaknesses to gain access, rights or privileges
	Authorisation violation →	An entity authorised for one resource uses that authority for unauthorised purpose
Underlying threats	Eavesdropping →	Information is revealed by monitoring communications: e.g., by EM/RF interception
	Traffic analysis →	Observation of communications traffic patterns
	Indiscretions by personnel →	An authorised person discloses information to an unauthorised person, reasons for which may include money, favours or carelessness

Figure 2.2. Examples of threat classification to contemporary computer networks (Tabulated from work done by Ford, 1994).

As may be seen from Figure 2.2, amongst Ford's underlying threats is that of eavesdropping, the use of which may enable the more fundamental threat of information leakage. The digital nature of serially transmitted data streams (such as the smart-card's communications link and RS-232 links) exhibits short rise and fall times which correspond to high frequency signal components that, in turn, yield considerable radiation for observation by an eavesdropper. The work of Smulders (1990), showed that reconstructed messages with bit error rates of approximately 0.01 (1%) could be achieved reliably, sensing at a distance of between one and seven metres when eavesdropping on *shielded* RS-232 cables. Now consider that the asynchronous serial transmissions being monitored tend to be grouped in characters of seven to eleven bits, simplifying the task of reconstructing any unknown bits by examination in context. Notably, the equipment used in these experiments was neither expensive nor sophisticated, comprising a standard AM/FM radio receiver, a simple whip antenna of one metre length and a hard limiter circuit to reconstruct the digital nature of the data. Smulders' article concludes: "the receiver and recording equipment for intercepting RS-232 data signals are very small, simple and cheap compared with the equipment needed for the interception of video signals. ... Because of this fact ... we have to take special account of the RS-232 eavesdropping possibilities in vulnerability studies." Although the smart-card's read/write machine would not be expected to possess long cable lengths and its voltage swings are less than those of RS-232, Smulders' work demonstrates that, in addition to the direct monitoring circuitry exemplified in Figure 1.2, serial link eavesdropping is possible using external monitoring circuitry.

Herskovitz (1995, June) points out the value of permitting and observing, rather than disrupting an enemy's communications. He cites as prime examples of this the gains provided by the Allies' observation of enciphered enemy messages during World War Two, after the enemy encryption mechanisms of Enigma and Purple had been cracked. Recognising the inevitability that a communicated message needs protection, Herskovitz (1995, June) outlines "four basic approaches to securing the information conveyed by a communication system...:

1) permitting no unintentional signal radiation - signals may be bounded by metallic waveguide or within an optical fibre;
2) minimising stray radiation ...;

3) making signal detection difficult - frequency, time or modulation coding;

4) permitting detection but denying understanding - enciphering."

Of these, 1) and 2) may be afforded by the smart-card manufacturer's use of *intrinsic security*, described in chapter one, with respect to the situation of components used in the card and read/write machine. Approach number 4) may be offered by *application security*, also described in chapter one, when software requirements are specified by the smart-card providers. Currently, no attempt is made by ISO-7816 compliant cards to provide approach number 3).

2.2 Other Literature of Significance to this Study

The field of military communications has advanced the topic of channel-hopping to a state of maturity where its methods and terminology are now well-defined. Torrieri (1994, p. 199) provides the following development of channel-hopping using carrier frequencies in his description of its fundamental concepts:

> "Frequency-hopping is the periodic changing of the frequency or frequency set associated with a transmission. A frequency-hopping signal may be regarded as a sequence of modulated pulses with pseudorandom carrier frequencies. The set of possible carrier frequencies is called the hopset. Hopping occurs over a frequency band that includes a number of frequency channels."

The effectiveness of channel-hopping depends upon the unpredictability of its hopping pattern. Hops may accompany each communicated data packet (fast hopping) or may be spread over several packets (slow hopping). Furthermore, hopping may occur across the bandwidth of a single channel, or across several disparate channels. To analogise the contents of this last statement with respect to smart-cards might be to hop using time-divisions on a link connected to a single card contact, or alternatively to hop between links connected to several of the card's contacts. A method of combining data to be communicated with a synthesised channel-hopping pattern to produce the channel-hopping signal is shown in Figure 2.3. Further security may be added to the channel-hopping pattern, for example, by combining a secret key with the time of day so that the pattern does not become recognisably repetitious over a prolonged period. This method implies that both key and time of day are correctly synchronised between sender and receiver before hopping may occur, and further implies that the key is changed infrequently relative to the volume of a link's data packets.

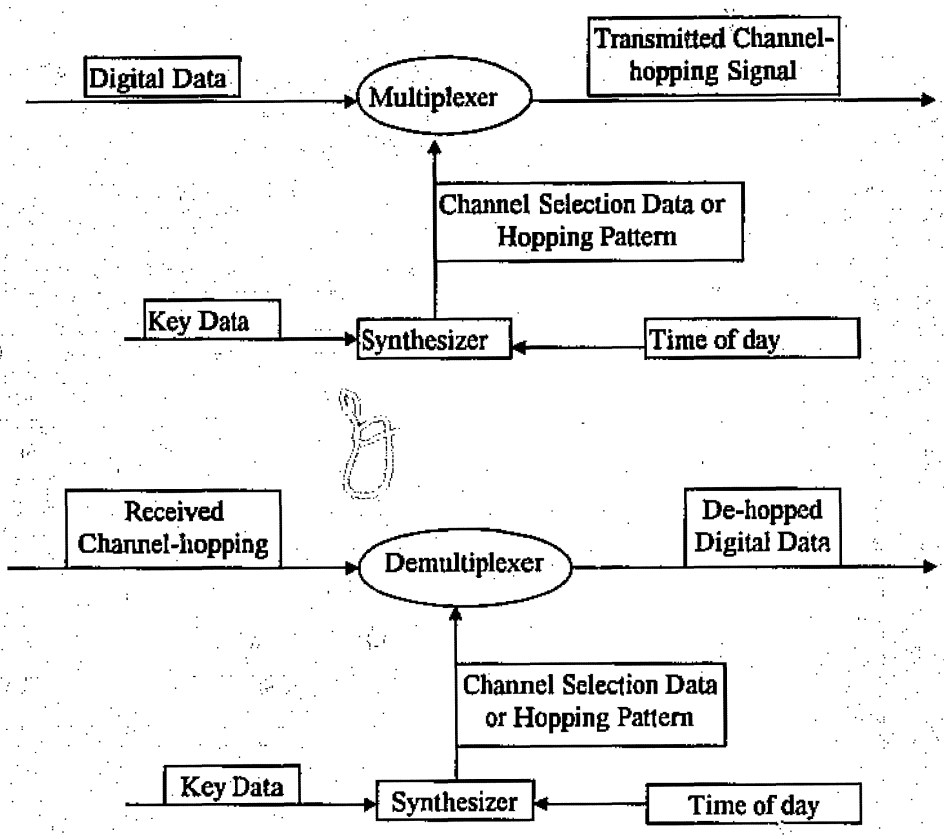

Figure 2.3. **General Form of Channel-hopping System. (Adapted from diagrams and descriptions in Torrieri 1994, pp. 200-204)**

Radar systems (themselves forming both transmitter and receiver of a communications system) use channel-hopping (Herskovitz, 1995, July), varying their carrier frequency to avoid their signals being jammed rather than to avoid eavesdropping. Jamming is also a threat to communication systems when, for example, robust encryption prevents an enemy making use of intercepted data. According to Greiner & Reissberg (1987) and Herskovitz (1995) transmitters must detect that they are being jammed, then reactively switch to the next hop in order to re-establish the link. This type of hopping is known as adaptive reaction channel-hopping where communicating parties, independently and "on the fly", assess the probability of the next channel to settle upon, rather than hopping according to a consistent pre-communicated pattern. Greiner & Reissberg (1987) report that adaptive reaction systems could deal with rates of two hops per second while providing low communication bit rates of approximately one hundred baud. However, when compared to standard hopping rates in excess of five hundred hops per second (Telefunken System Technik, 1993), it may be suggested that the computation effort required for channel selection reduces data throughput. Although Greiner & Reissberg's 1987 report may be considered "old", recall that the CPUs used in many of today's smart-cards were designed in the 1980s (Townend, 1995), implying similar performance to that found by Greiner & Reissberg. Thus, the author feels it is justifiable to consider the application of a pre-defined, pre-communicated hopping pattern.

As often happens, the technology devised for military purpose finds its way into everyday commercial use. Channel-hopping no longer exists exclusively within the domain of military communications. Dellecave (1995) reports that vendors making products to meet the specification of the forthcoming IEEE 802.11 standard for wireless Local Area Networks (LAN) have chosen carrier-frequency based channel-hopping techniques to improve reliability. In a report on the offerings of wireless LANs, Rigney (1995) suitably sums up channel-hopping's perceived advantage: "This spreading of the data protects the signal from eavesdropping and interference."

In addition to data, communications networks often provide a vehicle for their own packaged control signals. Verschuren, Govaerts and Vandewalle (1991) observe that the data which flows through a communication network is typically twofold: being,

- user data; and
- protocol control information (PCI) governing the communication between entities.

In the context of this study, the user data equates to the data presently sent from the smart-card to its read/write machine, and for which it is the study's objective to add protection. The PCI comprises details of the channel-hopping technique which will add the protection. This may be seen as the provision of a confidentiality service that prevents, for example, unauthorised disclosure of information from the communications link by a passive attack or eavesdropping. Verschuren et al. (1991), propose that such confidentiality protection should be considered at two levels or layers of the communication system: namely, the application layer and the physical layer, by the use of encipherment or cryptography.

Cryptography, seen by Parker (1983) as "The premier safeguard against computer crime", is defined by Vandewalle, Govaerts and Preneel (1991) as "the science of techniques which make data unintelligible and unmodifiable by outsiders...and still accessible or verifiable by the legitimate receiver." Some smart-cards provide facilities for encryption of data by the DES algorithm about which Vandewalle et al. (1991) state "Concerning the security of DES there is a rather general consensus among the cryptographic researchers that it is an extremely good algorithm with an unfortunately small key of 56 bits. Hence it is best used in a multi-encryption scheme (triple encryption with two keys)...." Later in their discussion Vanderwalle et al. (1991) acknowledge that speed of encryption and decryption is often too slow and they put forward a hybrid scheme. In this scheme, when speed is critical, key-exchange is effected under a strong encryption (e.g., RSA or multi-encryption of DES) and information exchange is effected under less onerous algorithms. Relating the scheme to the context of the smart-card to read/write machine communications link, the key exchange may equate to the channel-hopping pattern, transferred under strong encryption, whilst the ensuing information exchange is afforded the protection of the channel-hopping itself.

So far, the development of channel-hopping has considered that a message may be multiplexed across many channels, according to an acceptahle sequence which is either known to, or derivable by, the transmitter and legitimate receiver. If this was the totality of the channel-hopping implementation, then observation of all channels may reveal the valid one. An obvious improvement would be the concurrent transmission of both the valid message and random characters on some or all of the available channels. However, where all channels are being monitored, automated analysis which searches for and combines potential lexemes, may reveal the valid message. To obfuscate the situation, further misinformation may be offered to an interloper by means of "ambiguity, paradox and logic of recognition" (Ingleby, 1988). Broadly speaking, equally plausible message sequences are transmitted on some or all of those channels *not* being used to convey the real data. Ingleby's work is directed to speech and image recognition where, for example, phoneme strings are assessed for possible patterns to aid in the recognition of spoken messages. However, as Sebesta (1989) suggests, an automated language recogniser acts on syntactic units (called lexemes) which have been produced from a known alphabet within a known lexical specification. Consider the following text, " *'Twas brillig and the slimy toves did gyre and gymbal in the wabe*", from Lewis Carroll's *Jabberwocky*, which might be transmitted concurrently with a valid message. The transmitted material could be formed from combinations of the text and the valid message so as to fool an automated recognition tool scanning all received channels for lexeme possibilities. Such applied misinformation means that the eavesdropper must decide which is the valid message.

Having developed the requirements of the proposed method thus far, it is appropriate to identify the various forms of attack which it could be expected to endure. For this, the study looks toward attacks which cryptanalysts apply in their investigations of encrypted data. First, however, the common terminology is introduced and conventional methods are presented in overview. Longley, Dawson & Caelli's (1994) examination of traditional ciphers and coding techniques suggests that they may be regarded in the following manner: in general, algorithms used do not treat the whole volume of plaintext in one pass, and they break it up into smaller units for processing. *Ciphering* techniques combine these units with a key, resulting in the production of ciphertext, a transformation of the plaintext. Longley et al (1994) indicate that the data may be processed at a fundamental unit (of bit or byte) level, which is considered as a *stream cipher* method or, alternatively encrypted in blocks of several fundamental units at a time: which method is known as a *block cipher*. Stream ciphers are characterised by their application of simple algorithms where plaintext and key streams of identical length are combined to produce a stream of ciphertext identical in length to that of the plaintext. Block ciphers typically utilise more complex algorithms where the length of the applied key is the same as that of the block. The key may be applied symmetrically by both communicating parties or, asymmetrically, where the algorithm permits each party to apply a different key for their enciphering/deciphering operations. *Coding* provides protection of transmitted data where symbols in the plaintext are replaced by corresponding symbols, or groups of symbols, known only to the legitimate communicating parties. Importantly, the product of coding yields a stream, the length of which may differ markedly from that of the plaintext.

The above techniques' capability of resisting attacks depends upon the secrecy of or denial, to the cryptanalyst, of some knowledge employed in the protection method used. Attacks may assume:

- knowledge of the technique used, whereby security depends upon the secrecy of the key;
- that sufficient ciphertext is available for scrutiny;
- that some plaintext is known to be associated with captured ciphertext;
- that correspondence has been established between known plaintext and known ciphertext;
- a knowledge of the chosen language's redundancy, recognised by Shannon (1949), which may be exploited for statistical analysis.

Seberry and Pieprzk (1989) provide a description of common cipher techniques, together with suggested forms of cryptanalysis for each. Longley et al. (1994) summarise the most common forms of attack, which may be used in isolation or in combination, as:

- *key exhaustion*, where attempts using many keys are made to extract the valid message;
- *statistical analysis* where, for example, the known frequency of usage of letters in the chosen language is used to isolate probable instances of the valid message; and
- *mathematical analysis*, where the cryptanalyst tries their knowledge of algorithms against the collected ciphertext to regain the valid message.

2.3 Summary

The discussion in this chapter has sought to establish that, stemming from the recognition that a communications link inevitably will be monitored, protection measures need to be applied so that the fundamental threat of information disclosure does not occur. The discussion has further sought to establish that the proven military practice of channel-hopping, now entering more general use, may appropriately be applied to the smart-card's communications link(s), affording added protection. Hopping patterns may be generated from a combination of a secret key and, if available, a synchronised time-of-day clock, so as to minimise decryption attempts based upon the isolation of a repetitive key. Having surmised that "on-the-fly" or reactive channel selection may introduce an excessive burden on the smart-card's CPU, the concept of applying pre-arranged hopping patterns, initially communicated under strong encryption between card and read/write machine, has been put forward as suitable added protection for ensuing communicated data. Requirements for the method to be developed were concluded by proposing that applied misinformation, in the form of apparently meaningful messages transmitted concurrently to the valid one, may further mislead and complicate the eavesdropper's task, even when automatic recognition is applied. Finally, the nature of attacks to which the study's method would be subjected were identified, drawing upon the experiences of conventional cipher and coding techniques.

3. Research Design

The discussion in previous chapters suggests that data transmitted via the smart-card's communications link may be afforded some protection by the combination of a suitable channel-hopping mechanism and the co-transmission of applied misinformation. With regard to the eight-bit processors used in smart-cards the fundamental aim, then, is to surround the valid message with plausible misinformation, accomplished in a manner which is economical in terms of machine resources. By plausible it is meant that the misinformation will match as closely as possible the nature and type of the valid message. The co-transmission of such misinformation reduces an eavesdropper's advantage of using automated lexical analysis to retrieve the valid message from the apparently randomly shuffled transmitted text. While providing a measure of security superior to that of transmitting data as plain-text, the method should allow the smart-card processor to offer an option which, albeit less secure than encryption, does not require encryption's attendant computational overheads.

This chapter provides a description of the basic method implemented and of enhancements to that basic method evaluated in this study.

3.1 General Method

For the purpose of the study, the valid messages, embedded within their accompanying misinformation, arrive at the legitimate receiver via available channels, as a result of the following distinct processes and considerations:

- creation of the channel-hopping pattern;
- communication of the channel-hopping pattern to be used in the forthcoming communications session, in a secure manner, to the legitimate recipient;
- selection of appropriate misinformation to transmit concurrently with the valid message;
- storage of, and subsequent access to, the misinformation data; and
- transmission of the messages, with valid data according to the pre-arranged channel-hopping pattern, and with misinformation data co-transmitted on the remaining channels.

Of the above list, it is to be noted that the first four items, which are performed *once* per communications session, are concerned primarily with the organisation of the transmitted data. This leaves the processor little additional work to perform at the time of transmission. The situation is similar for the receiver: in order to isolate the valid message in real-time, the legitimate receiver has only to select, according to the pre-arranged channel-hopping pattern, the valid characters from those received. In other words, by following the pattern, the receiver is able to reject received misinformation characters as they arrive. By contrast, encrypted data must be submitted to an encryption process prior to transmission, and a decryption process before the valid message may be reclaimed from the received data. These resource consuming preparations accompany *each* transmitted message during a communications session.

3.2 Specific Procedures

3.2.1 Creation of the channel-hopping pattern

Recall from chapter two that hopping may occur across the bandwidth of a single channel, or across several disparate channels. Analogies to these, in the context of smart-cards, are that hopping may be accomplished:

- using time-divisions on a link connected to a single smart-card contact, whereby several logical channels are mapped onto the single physical channel of the smart-card's communications link; or alternatively
- by hopping between links connected to several of the smart-card's contacts, whereby several logical channels are mapped onto several physical links between smart-card and its read/write machine.

It is necessary to decide how many channels will be implemented within the physical links available. The mapping of logical to physical channels for the pilot implementation is according to the reasoning and practical observations which follow.

- Smart-card hardware is unlikely to support the use of more than the two designated input/output (I/O) lines within the eight available.
- Smart-cards which conform to ISO-7816, i.e., those in widespread use, normally communicate via one I/O line;
- While the mapping of many logical channels to one I/O line's physical channel will deteriorate the useful bit rate of that channel, it represents a facile mechanism by which to demonstrate the concepts embodied in the study. Furthermore, recalling that the bulk of the computation is performed prior to transmission, such a mapping represents the worst-case situation for the channel-hopping mechanism under development. Put another way, if one accepts that the organisational work is accomplished largely prior to transmission, the availability of extra physical channels will result in an increase of data throughput approximately proportional to the ratio of the increased number of channels.

The example in Figure 3.1 shows seven logical channels, meaning that every valid character (V) is accompanied by six misinformation characters (*M*).

Figure 3.1. Example of seven channels. In this case, the valid message character occupies the fourth position.

There is a need to use a fresh hopping pattern for each communication session between card and read/write machine, thus minimising repetition of pattern usage. Further, to minimise repetition, the pattern's length should be sufficient to disguise the messages which need to be transferred. That is to say, longer messages need a longer pattern than relatively short messages to eschew repetition throughout transmission. The pattern may be filled by repeated use of a pseudorandom number generator operating within the range of available channels. Alternatively, where a pseudorandom number generator proves too cyclical or produces weak patterns, the hopping pattern may be selected from tables of suitable, previously generated, patterns. An example of a hopping pattern is given in Figure 3.2.

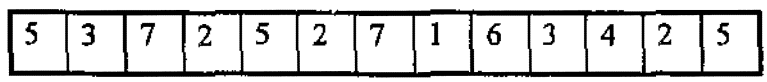

Figure 3.2. Example of hopping pattern of length 13 for seven channels.

Using the examples provided in Figures 3.1 and 3.2, each cycle of the combination of Valid (V) and Misinformation (*M*) characters will be transmitted as given in Figure 3.3. If the valid message is longer than the pattern, the cycle will be repeated.

Channel →	0	1	2	3	4	5	6

Time							
	M	*M*	*M*	*M*	**V**	*M*	*M*
↓	*M*	*M*	**V**	*M*	*M*	*M*	*M*
↓	*M*	*M*	*M*	*M*	*M*	*M*	**V**
↓	*M*	**V**	*M*	*M*	*M*	*M*	*M*
↓	*M*	*M*	*M*	*M*	**V**	*M*	*M*
↓	*M*	**V**	*M*	*M*	*M*	*M*	*M*
↓	*M*	*M*	*M*	*M*	*M*	*M*	**V**
↓	**V**	*M*	*M*	*M*	*M*	*M*	*M*
↓	*M*	*M*	*M*	*M*	*M*	**V**	*M*
↓	*M*	*M*	**V**	*M*	*M*	*M*	*M*
↓	*M*	*M*	*M*	**V**	*M*	*M*	*M*
↓	*M*	**V**	*M*	*M*	*M*	*M*	*M*
↓	*M*	*M*	*M*	*M*	**V**	*M*	*M*

Figure 3.3. Cycle showing hopping pattern of length 13 over 7 channels, illustrating the apparent randomised shuffling of the valid data.

3.2.2 Communication of the channel-hopping pattern to the legitimate receiver

To minimise detection where some plaintext is known by an eavesdropper, a fresh hopping pattern is generated for each session between the smart-card and its read/write machine. To obstruct unwanted eavesdroppers, the hopping pattern list (effectively the "key" to the process) must be communicated under great secrecy. Whilst it is not intended to include key management within the scope of this study, the following guidelines are suggested for communication of the hopping pattern to the legitimate receiver.

Many smart-cards provide on-card encryption routines, typically that of DES/DEA. The application of triple DES is acknowledged, by Burton, Kaliski & Robshaw (1996), to provide high levels of security. The hopping pattern may be communicated under such encryption, making use of the on-card DES routines and associated DES key management facilities. Where such on-card facilities do not exist, a DES implementation such as that provided by Brown in Seberry and Pieprzyk (1989) may be utilised, and key management techniques, as described by Caelli, Dawson and Longley (1994) for symmetric ciphers, may be applied.

For the purpose of the pilot implementation, it is assumed that each legitimate communicating party (i.e., the smart-card and its associated read/write machine) is aware of the channel-hopping list and that communication of the list has been conducted in a secure manner.

Clearly, key management may be a relatively time-consuming task for the smart-card. It is suggested that the generation and encryption of the one-off hopping pattern and its subsequent communication to the legitimate receiver may be accomplished shortly after the smart-card is inserted into its read/write machine. Notably, cardholders are accustomed to delays of several seconds at this "initialisation" phase of their anticipated transaction. Accordingly, to hold the cardholder's attention during any added delay, the card read/write machine might display a suitable promotional/welcome message.

As an alternative to the enciphered transmission of the complete hopping pattern list, it may be considered worthwhile to take advantage of the algorithmic nature of computer pseudo-random number generators by enciphering and communicating the seed with which the generator should be commenced, and the hopping pattern length. Where the two legitimate communicating parties use the same pseudo-random number generator, the hopping pattern may then be derived by each party applying the same seed.

3.2.3 Selection of appropriate misinformation for co-transmission

The conventional application of channel-hopping, as in military radio communications, operates across different carrier frequencies, taking advantage of a situation where it may be difficult to monitor all possible frequencies simultaneously. This study's adaptation of channel-hopping differs markedly in that all channels may be monitored with equal ease on the serial link. There is, therefore, a need for the application to surround the valid data with misinformation in which the valid data will appear inconspicuous. By example, it is appropriate to accompany valid messages of binary/textual/numerical data with binary/textual/numerical misinformation respectively.

There is clearly a need to provide/generate misinformation which will camouflage all possible types of valid data during transmission. One means of satisfying this need is to provide prepared libraries of misinformation from which to choose before transmission. Smart-card application programmers may reasonably be expected to know and/or control the type of data communicated by their applications. Thus, when writing their applications they may be able to select, from the various types of supplied misinformation, the type(s) most appropriate for co-transmission with their valid message.

3.2.4 Storage of, and subsequent access to, the misinformation data

Smart-card MCUs notoriously provide little RAM: typically 128, 256, 512 or 640 bytes. However, also typically, they do provide between 8 and 20 kilobytes of ROM in which application code and fixed data may reside. These characteristics mandate an approach which is sparing in its use of RAM, but which may reasonably allocate sufficient of the on-chip ROM for its purpose.

Look-up tables provide a means of rapidly accessing data, and for the pilot implementation a small database is constructed to contain the misinformation for transmission in non-valid message channels. Examples of textual misinformation are given in Figure 3.4, and these are, in fact, accessed from their ROM storage as arrays of null terminated strings. Misinformation accompanying numeric or digital data may either be generated in a pseudorandom fashion at the point of transmission, or libraries of prepared misinformation may be constructed to be held in ROM.

/*string */	/* of the misinformation in which the valid text will be hidden*/
char *misinfo1	*"Twas brillig and the slithy toves did gyre and gymbal in the wabe: All mimsy were the borogoves and the mome raths outgrabe ";*
char *misinfo2	*"Beware the Jabberwock, my son! The jaws that bite, the claws that catch! Beware the Jubjub bird, and shun The frumious Bandersnatch! ";*
char *misinfo3	*"He took his vorpal sword in hand: Long time the manxome foe he sought, so rested by the Tumtum tree, And stood awhile in thought. ";*
char *misinfo4	*"And as in uffish thought he stood, The Jabberwock, with eyes of flame came whiffing through the tulgey wood, and burbled as it came! ";*
char *misinfo5	*"One, two! One, two! And through and through, the vorpal blade went snicker snack! He left it dead, and with its head, he went galumping back ";*
Char *misinfo6	*"And hast thou slain the Jabberwock? Come to my arms, my beamish boy! O frabjous day! Callooh! Callay! He chortled in his joy. ";*

Figure 3.4. Examples of textual misinformation. Quoted from Lewis Carrol's Jabberwocky.

The study draws, in part, upon an adaptation of the techniques devised by Ingleby (1988) for the introduction of ambiguity into a transmitted message. The ambiguity is introduced by the concurrent transmission of appropriate misinformation and thus may lead to confusion of an eavesdropper's logic of recognition. In keeping with Ingleby's examples the author has quoted and used Carrol's works for the purposely designed ambiguity they contain. For the reader's information Lewis Carrol was the pen name of Charles Lutwidge Dodgson (1832-1898), a mathematician who, according to

Everyman's Encyclopaedia (1973), studied and subsequently lectured at Oxford University between 1854 and 1881. Amongst his publications were mathematical papers and several works of fiction.

Transmission of the message

Having completed the previous steps, the valid message may now be transmitted using a hopping pattern such as in Figure 3.3. Assuming a textual message is desired to be co-transmitted with misinformation such as is illustrated in Figure 3.4, the following simple algorithm may be applied.

```
initialise the variables used;
for (each character in the valid message) loop
  for (each channel) loop
    if ( this channel = this hopping pattern)  then
      transmit next character from valid message;
    else
     transmit next character from randomly selected misinformation string;
    end if;
  end loop;
  advance hopping pattern in a cyclic fashion;
end loop;
```

Using the above algorithm, channels are filled with characters as follows. The channel coinciding with the current element of the hopping pattern is allocated the next character from the valid message. Note that each remaining channel is supplied with the next character from a pseudo-randomly selected misinformation string/datastore. This prevents two identical valid messages being co-transmitted with identical misinformation, further increasing the apparent randomness of the production.

The algorithm for retrieving the valid message from the stream of received characters is as follows:

```
initialise the variables used;
while (characters are arriving) loop
  for (each channel) loop
    if (this channel = this hopping pattern) then
      add next received character to received message;
    else
      discard next received character;
    end if;
  end loop;
end loop;
```

3.3 Potential Enhancements

Having established the functionality of the basic form of channel-hopping coupled with applied misinformation, the following potential enhancements, intended to minimise repetition, are considered:

- addition of a time-of-day-clock modifier (see Figure 2.3) so that identical valid messages transmitted at different times cannot have the same hopping pattern;
- maximise usage of the set of misinformation by swapping applicable consonants (or groups of consonants) contained in the misinformation at the moment of transmission;
- random changing of the bit order of characters transmitted, i.e., establishing a pattern for hopping bits to be superimposed upon the hopping pattern applied to the characters on the link. If this were to be effected upon all valid characters, it is possible that the entire message may be camouflaged within binary data.

The above listed enhancements will be discussed in turn.

3.3.1 Addition of a time-of-day-clock modifier

As illustrated in Figure 2.3, the time of day clock modifier may be used to prevent identical valid messages being transmitted with the same hopping pattern. In order to implement a reliable clock, however, generally requires an interrupt driven timer, a feature which is absent from typical smart-card CPUs. Accordingly, this enhancement was seen as impractical to realise within the context of a smart-card and was not pursued.

3.3.2 Maximise usage of the set of misinformation by swapping applicable consonants at the moment of transmission.

Possibilities for implementing such swapping include:

- selections employing a pseudorandom derived swap; and
- selection of exchangeable characters by reference to their normal frequency of occurrence in the current language.

Exemplification of the effect of consonant swapping is to be found in Lewis Carrol's Jabberwocky in the word "*slithy*". Carrol's example shows that an '*m*' may have been exchanged with the pair "*th*" to present a word which at first sight appears English-like. Note that similar effect might have been achieved by replacing the '*m*' with '*b*' (*sliby*), '*d*' (*slidy*), and so on. Importantly, a lexical analyser would probably accept these as plausible words. This enhancement was deemed to offer improvement and was implemented in the study. The modified transmission algorithm appears below, with modifications appearing in bold text.

```
initialise  the variables used;
for (each character in the valid message) loop
  for (each channel) loop
    if ( this channel = this hopping pattern)  then
      transmit next character from valid message;
    else
     get next character from randomly selected misinformation string;
     if (it is a consonant) then
       randomly swap it with another consonant;
     end if;
     transmit resultant character;
    end if;
  end loop;
  advance hopping pattern in a cyclic fashion;
end loop;
```

Note that this enhancement requires no further processing to be accomplished by the legitimate receiver.

For simplicity, consonant swapping was limited to that of single character and all consonants, excepting those deemed inappropriate for swapping (i.e., [q,w,x]), were considered as equal candidates for swapping.

3.3.3 Random changing of the bit order of characters transmitted

As an enhancement, such a technique offers two potential advantages: all transmitted data now becomes binary in nature, removing the need for such carefully selected misinformation to be applied to enveloping the valid message; and, it increases the complexity of the eavesdropper's task.

Disadvantages may also accompany the method, however, dependant upon the level of granularity of bit re-arrangement imposed upon the transmitted data. Where bit re-arrangement is effected at character level, it may be shown to be a relatively facile task to perform an automated search of all possible bit combinations. All or part of the intercepted message may be examined in this way, possibly leading to recovery of the bit re-arrangement pattern. Where bit re-arrangement is effected on a wider scope, e.g., the entire message, exhaustive searches become more time-consuming, but the demands upon sender and legitimate recipient rise accordingly. Notably, the incorporation of this technique, with bit-order rearrangement effected on a wide scope, can be seen to deviate from the original operation where the organisational work is acccomplished ahead of the transmission, and where little subsequent work is required of the receiver to extract the valid message.

As discussed previously, an application programmer may appropriately select the nature of misinformation for co-transmission. In like manner, the decision to apply additional bit re-arrangement, with its attendant CPU resource costs, could be left to the application programmer's discretion.

The algorithm required to implement bit re-arrangement, at character level, is given below, with those modifications required to augment the basic method illustrated in bold text.

```
initialise the variables used;
for (each character in the valid message) loop
  for (each channel) loop
    if ( this channel = this hopping pattern) then
      get next character from valid message;
    else
      get next character from randomly selected misinformation string;
    end if;
    re-arrange next character's bit order according to next bit-order pattern;
    transmit resultant character;
  end loop;
  advance hopping pattern in a cyclic fashion;
end loop;
```

The algorithm for the legitimate receiver is given below, again with modifications highlighted in bold text:

```
initialise the variables used;
while (characters are arriving) loop
  for each channel loop
    if (this channel = this hopping pattern) then
      get next received character
      re-arrange received character's bit order according to next bit-order pattern;
      add to received message;
    else
      discard next received character;
    end if;
  end loop;
end loop;
```

3.4 Summary

The components of design for the pilot implementation whereby transmitted data may be protected by shuffling it across several logical channels in randomised fashion, and enveloping it within applied plausible misinformation, have been described.

The processes and considerations contributing to the communication of the protected data were developed to yield the basic design. During the production of the basic design, potential improvements were identified. These enhancements to the basic design have been discussed and evaluated for inclusion in this implementation or future studies. Scenarios of the basic design and the adopted enhancements have been presented with their resultant algorithms. The design, espousing economy of implementation, is respectful of the basic level of facilities offered by smart-card CPUs.

4. Implementation and Findings

Recall, from the previous chapter, that our fundamental aim is to intersperse the valid message within a cocoon of plausible misinformation, in order to confuse a potential eavesdropper. This concealment is to be accomplished in a manner which is economical in terms of the machine resources available from an eight-bit CPU exhibiting features similar to those found in contemporary smart-cards. Design decisions are required for the selection of appropriate software tools to perform the specific components of the implementation, and for the choice of a suitable target system which is adequately equipped to produce a test program illustrating the scenarios previously identified.

4.1 Selection of the software tools and target system

For reasons of availability, economy, and its feature-similarity to typical smart-card CPUs, a proprietary Motorola evaluation system centred on Motorola's 68HC11 (MC68HC11) eight-bit microcontroller was employed as a target system. Usage of the evaluation system's hardware was constrained, for the study's purpose, to those facilities normally found within a smart-card CPU. The constraints precluded the use of interrupts (e.g., for timing purposes) and/or the on-chip serial ports for data transmission.

Similar reasons prompted the initial use of the ImageCraft C cross-compiler/assembler for the MC68HC11, as the vehicle for code generation. The ImageCraft C cross-compiler, however, proved to be incomplete in its support for the ANSI C standard. Limitations caused by such non-compliance prompted the employment of a Hi-Tech cross compiler/assembler for the later stages of implementation. Notably, neither the ImageCraft nor the Hi-Tech assembler was found to be fully compliant with Motorola's assembly language syntax, particularly in the support of macros, although this has not impacted upon the implementation executables.

Timing measurements were made using a Hewlett Packard Model HP54600A one hundred MHz digital storage oscilloscope, Serial Number 3334031110, supplied by Edith Cowan University.

4.2 Implementation of the study

The design discussed in chapter three suggests three scenarios to be implemented:

- the transmission of the valid message, shuffled and encapsulated within selected applied misinformation - this is labelled the *basic method*;
- the basic method, enhanced by the random substitution of consonants contained within the applied misinformation, thus maximising re-use of the misinformation set - this is labelled the *substitution method*; and
- the basic method, augmented by the random re-arranging/scrambling of the bit order of transmitted characters - this is labelled the *scrambled method.*

It may be seen that variations and combinations of the above scenarios may be employed to gain further improvement. However, for the purpose of the study, implementation has been limited to these scenarios.

The implementation concerning each of these will now be presented in turn, with every one being discussed in terms of:

- its requisite implementation code in the C language;
- the test data chosen to demonstrate facilities offered by the technique; and
- the resultant production given by the application of the scenario's randomised shuffle together with its associated applied misinformation.

Timing measurements, yielding a measure of CPU resource usage, observed for each scenario will also be presented for differing data-lengths. Annotated source code listings may be found in Appendix B.

4.2.1 The basic method

It is appropriate to view the code of the basic method, in the 68HC11 target system, as providing two interfaced layers: namely,

- fundamental serial communications support; and
- the Randomised-Shuffle implementation.

In keeping with smart-card technology, the asynchronous serial communications implemented for this study are effected by the precisely-timed toggling on/off of a bit connected to one of the target MCU's Input/Output (I/O) ports. Synchronisation is necessary between communicating devices and, by convention, those employing asynchronous serial techniques align themselves using the transitions of each character's start bit. Once the two parties have become synchronised, each successive bit of the communicated character is sampled with reference to the period elapsed from the character synchronisation point. Clearly, for this technique to work, the jitter (the variation from the anticipated sample time) for each bit transition must not be excessive. Such exactitude in timing mandated the use of assembly language for the serial communications support layer. A simple function-oriented interface, incorporating register-based parameter passing, is presented to the Randomised-Shuffle layer. The serial communications support layer implementation remains constant for all three scenarios, and its source code was supplied by a third party (see Acknowledgements). This code plays no part other than providing the passage of characters to/from the target system in timely manner and would, in any case, be already present in the smart-card's existing software. As a consequence of this, the serial communications support layer is discussed no further.

During testing, it became apparent that the Pseudo-Random Number Generation (PRNG) function, called *rand( )*, supplied with the C compiler introduced considerable delays (approximately one-half a character's transmission time) when used to select the channel for each misinformation character to be transmitted. This PRNG acts upon, and furnishes as a result, a thirty-two bit integer. Whilst today's contemporary thirty-two bit CPUs provide hardware rapid multiplication and addition instructions, facilitating similar linear congruential thirty-two bit calculations on smart-card eight-bit

CPUs takes considerably longer to perform. Furthermore, such a linear congruential method, implementing an affine relationship based upon two large prime constants, is difficult to separate into discrete manageable components. However, there exists another category of PRNG which uses a shift register. For the purpose of this study, this type of PRNG is attractive as its register may be shifted during the idle periods that a smart-card's CPU endures during bit transmission of characters. Importantly, such a shift register mechanism operates at a fundamental level of one bit at a time: thus, permitting the interweaving of the bit-shifts within several of the aforementioned discrete idle periods. Put another way, concurrent with the transmission of a character, the CPU may usefully be occupied in preparing the next pseudo-random number. Scheier (1992) proposes a bit-shift PRNG mechanism whereby the incoming bit's state is determined by inspection of bits according to a polynomial of $2^{31} + 2^6 + 2^4 + 2^2 + 2^1 + 2^0$, for which the C code is presented in Figure 4.1. An explanation of the effective and optimised use of polynomials within bit-shift mechanisms, albeit for cyclic redundancy check applications, is provided in Kientzle (1997).

```
static unsigned long random_seed = 0x12736781L;

int random_bit( void)
{
 random_seed =
 (((
    (random_seed >> 31) //XOR with bit 31
   ^(random_seed >> 6 ) //XOR with bit 6
   ^(random_seed >> 4) //XOR with bit 4
   ^(random_seed >> 2) //XOR with bit 2
   ^(random_seed >> 1) //XOR with bit 1
   ^(random_seed)  //XOR with bit 0
  & 0x00000001)  //isolate resultant bit 0
  << 31)  //align resultant bit with bit 31
  | (random_seed >> 1)); //and move it back into bit 31 of random_seed

  return random_seed & 0x00000001;  //then, return the "random"-bit
}
```

Figure 4.1. **C code to implement Scheier's (1992) polynomial-based shift register PRNG.**

Notably, Scheier's mechanism inspects (via an exclusive-OR) the state of six bits of the thirty-two bit register *random_seed*, lending itself well to optimisation for a CPU with only eight bit registers. The author's highly optimised assembly code for this appears in Appendix B. Interwoven within the ten bits (eight data bits, plus one parity and one stop bit) of each transmitted character, the code was found to be rapid enough to generate a thirty-two bit pseudo-random number. As the basic method requires only three-bit pseudo-random numbers for each misinformation character's channel selection, there remains a redundancy sufficient to support transmission rates many times higher than the ninety-six hundred b.p.s. used in the study.

The basic Randomised-Shuffle implementation, for convenience, was implemented using the C language. For the reader's convenience, the pseudo-code for the basic method appears in Figures 4.2 and 4.3.

```
initialise the variables used;
for (each character in the valid message) loop
  for (each channel) loop
    if (this channel = this hopping pattern)  then
      transmit next character from valid message;
    else
      transmit next character from randomly selected misinformation string;
    end if;
  end loop;
  advance hopping pattern in a cyclic fashion;
end loop;
```

Figure 4.2 **Pseudo-code for the basic Randomised-Shuffle method (Transmission).**

```
initialise the variables used;
while (characters are arriving) loop
  for (each channel) loop
    if (this channel = this hopping pattern) then
      add next received character to received message;
    else
      discard next received character;
    end if;
  end loop;
end loop;
```

Figure 4.3 **Pseudo-code for the basic Randomised-Shuffle method (Reception).**

4.2.1.1 Implementation code for the basic method

It is assumed that any variables external to the subprogram(s) may be deemed to be in an initialised and usable state prior to the subprogram(s) being invoked. Relevant pseudo-code, entered in comment form within the ensuing C code extracts, has been italicised and boldened for clarity.

The functionality of the scenario's implementation has been encapsulated within a C function entitled "*put_combined_message*", featured in Figure 4.4.

```
//Subprogram    : put_combined_message
//purpose       : sends the valid char, together with its
//              : surrounding misinformation characters
//              : out of the serial port I/O link
//return        : none
//parameters    : the_valid_message: points to start of valid message
//side-effects  : none

void put_combined_message( char * the_valid_message)
{int  this_hopping_pattern = 0,
      the_length = strlen(the_valid_message),
      current_char;

 //for (each character in the valid message) loop
 for ( current_char=0; current_char < the_length; current_char++)
 {int this_channel;
  //zero_channels_already_used
  memset( channels_already_used, 0, MAX_CHANNEL-1);

  // for (each channel) loop...
  for (this_channel=0; this_channel < MAX_CHANNEL; this_channel++)
  { //if(this_channel = this hopping pattern) then...
   if (this_channel == hopping_pattern[this_hopping_pattern])
   { //transmit next character from valid message
     put(the_valid_message[current_char]);
   }
   else
   { //transmit next character from randomly selected misinformation string
     put( get_next_misinfo_char( ));
   }//if
  }//for

  //advance hopping pattern in a cyclic fashion
  if(++this_hopping_pattern >= MAX_PATTERN)
   this_hopping_pattern = 0;
 }//for
}
```

Figure 4.4 **Subprogram to transmit the combined valid message and misinformation.**

It may be seen, from the code in Figure 4.4, that the subprogram *put_combined_message* relies upon data held in an external array *'hopping_pattern[]'* and a further subprogram *get_next_misinfo_char*. The former, as its name alludes, contains the hopping pattern understood by both transmitter and legitimate recipient, and the latter is presented in Figure 4.5.

```
//Subprogram     : get_next_misinfo_char
//purpose        : gets next character from randomly selected
//               : misinformation string.
//return         : the next misinformation char ( typed to byte)
//parameters     : none
//side-effects   : The array 'channels_already_used[]' has the appropriate entry updated,
//               : signifying which misinformation strings have already been used
//               : during the current cycle of transmitting an
//               : enveloped valid character.
//
byte get_next_misinfo_char( )
{//randomly choose a number from within the range of available channels
 int misinfo_selected = next_rand % (MAX_CHANNEL-1);

 //if this channel has been previously used for accompanying
 //the valid channel, then choose the first available 'free' channel
 while (channels_already_used [misinfo_selected] )
 {misinfo_selected += 1;
  if (misinfo_selected >= (MAX_CHANNEL-1))
    misinfo_selected = 0;
 }//while

 //record the usage of this channel
 channels_already_used[misinfo_selected] =1;

 //before returning with the next char from this channel
 return (next_char_from(misinfo_selected));
}
```

Figure 4.5. Implementation code for obtaining the next misinformation character within one hopping cycle.

Referring to Figure 4.5 it can be seen that, in addition to choosing randomly a misinformation channel (whose next character will accompany the character from the valid message) it is necessary to record that the channel has been used. In the case that the channel has already been used, the next available channel is selected in sequential fashion. The array *channels_already_used[]* records usage of misinformation characters for the cycle of transmitting a complete set of misinformation characters with each successive character from the valid message. Finally, subprogram *get_next_misinfo_char* is dependent upon the function *next_char_from* (Figure 4.6) which, as its name suggests, gets the next character from the misinformation channel selected. The function *next_char_from* relies upon the array *misinfo_pos[]* to record the currently available position of the next misinformation character within each of the misinformation strings.

```
//Subprogram     : next_char_from
//purpose        : gets the next misinformation character to
//               : be transmitted, selected from the available
//               : misinformation strings
//return         : next misinformation character
//parameters     : misinfo_selected (in mode): identifies the
//               : misinformation string from which the char
//               : is to be taken
//side-effects   : updates misinfo_pos[], a record of the last
//               : char used from each misinformation string
//
char next_char_from(int misinfo_selected)
{char next_misinfo_char;
 int index;

 //retrieve position of the next available character from this channel
 index = misinfo_pos[misinfo_selected];

 //obtain the character indexed by this position
 next_misinfo_char =  misinfo_channels[misinfo_selected][index];

 //record next available position
 index+=1;
 //applying bounds so as not to index outside the misinfo string
 if (index >= misinfo_len[misinfo_selected])
 {index = 0;
 }
 //record this updated index
 misinfo_pos[misinfo_selected] = index;

 return next_misinfo_char;
}
```

Figure 4.6 **Retrieving the next available misinformation character, then updating indexing information variables.**

4.2.1.2 Test data and resultant productions.

Two differing sets of test data are presented, together with their output from the implementation for selections of applied misinformation. In each case the microcontroller has been programmed to transmit first the hopping pattern (generated using the C compiler's pseudo-random number generator) and, second, the valid message, purely as an aid to clarity of demonstration.

Figure 4.7 shows a batch of textual data subjected to the randomised shuffle and accompanied by the textual misinformation featured in Figure 3.4. It may be seen that the valid message, a complete verse taken from Lewis Carroll's "*The Walrus and the Carpenter*", blends into its envelope of accompanying misinformation.

The pattern is: 2325335611003

The message is: The sun was shining on the sea, shining with all his might: he did his very best to make the billows smooth and bright and this was odd, because it was the middle of the night.

The output is: HATTOABwenhnendwee das at , hsoraeatubossr knwtoi ti lw hlnhlat eisu s hi ogJO fu asvfniehao bsbrins,hlndep ratiawitnwlthnhgo o!o cs ewu t ko,noghAsh In retdd imth y Jt thae ihes nbyh b s rotneehso!taroauogv,w n do ohes Tc:sd,ak h h?ideL n noTid injh dn Ctaego hggw yr mstJ iawoermbiu ethebg ttaaoh eh, t ntrhwmd a o tbyelil cgh a itemkhm,er aibn,vm wxoss lti eor, hpm mtmeiaiehn Igycf he yobtlbtehaec:lew saa hh sdmeeoiw e fat s dhwh bs oaifeentd:bul g atoyAhmch elslait ,snltc i chsaOcvmmok ileerfm eryr rsB eaew yb wshsbwateinjresfoaereutdfck s ie n! ttb ghodtyh ea et tHmy!Jheh arkeu boeClb o ujeTartugfuolmlthgbh oe t otoi bvuhebtimhr e lisdt l drtelCa,euan oadaeldlwls,g n, te adsh Ayaym sn!enh do dwu oo Hmweotsontdhi mTotce ,o hah hrednao nira d fldatt bhslrwsrhbe uimduih gile robaohieututld ne, gsi ra hndB i hnastbdaaen h s ej d wtoTeeouihniywrtgss.hta ntc swaA.g ab taanmldsrcHeu i lhe lomhl tAapld noidsBigen,dto kg w ata abrenhsbhocda eia usic uk n tt hhs vsolceOuerna f fiisJpean,aitli s b ltbtwt hhhs eawrtwoeyso h! wttooJr hcdu aoe gvObken, bihetmnes m,ri hhd wdye aotdidsnl cw osekdotgn:? ! o yloAfoLr n eoCd t ,Tdonhahg me eenTt t hd hr tnjieagimo o euJwgims hmgaty tbhbhlb .taeehae

Figure 4.7. Textual valid data enveloped within textual misinformation. A hopping pattern length of 13 was distributed across 7 channels.

The pattern is: 2325335611003
The message is: 0123456789012345678901234567890123456789012345678901234567890123456789
The output is:
HA0TOABwen1nendw2e das at 3, h4oracat5bossr k6wtoi ti7!8 hlnhl9t ei0u s hi1ogJO fu
a2vfnie3ao bsbr4ns,hl5dep rat6awit7wlthnh8o o!o cs9ewu t k0,1oghAsh2ln re3dd imt4 y Jt tha5
ihes6nbyh b 7 rotn8ehsoltaro9uogv0w n do1ohes Tc:2d,ak h 3?4deL n 5oTid 6njh d7 Ctaego
h8gw yr9mstJ ia0oermb1u ethebg 2taao3 eh, t4ntrhwmd 5 o tbye6i7 cgh a8itemk9m,er a0bn,vm
wxo1s lti2eor, hp3 mtme4aiehn l5ycf 6e yob7lbtehaee8lew saa9

Figure 4.8. Numeric valid data enveloped within textual misinformation. A hopping pattern length of 13 was distributed across 7 channels.

The set of test data depicted in Figure 4.8, shows the results of numeric data, subjected to the randomised shuffle technique, and enveloped within the misinformation illustrated in Figure 3.4. It is evident that the numeric data is conspicuous against the surrounding misinformation, emphasising the need for an appropriate choice of data with which to camouflage the valid message.

The pattern is: 2325335611003
The message is: 0123456789012345678901234567890123456789012345678901234567890123456789
The output is:
650422425114645725748619233100348499985567375596630503774852417491171103916911
166632361260973308943304785575269316945628677589011876663836930997880910958512
096333262539401938386753153763192213780688813939525291592050353217406637821909
154394711299516802641175476903371888809649666178805998711344188442469539180244
353879525112819961747411580045694975250067258365127673265239323443984203947847
555106128187794696272882904549

Figure 4.9. Numeric valid data enveloped within numeric misinformation. A hopping pattern length of 13 was distributed across 7 channels.

The same numeric data, this time enveloped in numeric data, is shown in Figure 4.9 and it may be seen that the valid data blends absolutely into the surrounding misinformation.

4.2.1.3 Timing Data

Observations for timing of application of the basic method for various message lengths are recorded in Figure 4.10. The times were measured using a hopping pattern of length thirteen, time division multiplexed across seven logical channels, within the bandwidth provided by one physical channel at a bit rate of ninety-six hundred bits per second (b.p.s.). Each character's transmission is performed according to a protocol of eight data bits, one stop bit and an even parity bit and, including the time necessary for setting up the simulated smart-card's I/O link, consumed 1.35ms.

Valid Message Length (characters)	Method Setup Time (ms)	Process only of Transmitter (ms)	Process only of Receiver (ms)	Process + Transmission (ms)
1	24	4.5	0.32	14
8	24	40	2.6	119
16	24	69	4.7	223
24	24	109	7.2	346

Figure 4.10. Timing data observed for the basic method of Randomised-Shuffle and Applied Misinformation.

Note that the transmission time includes the transfer of the number of valid characters times the number of channels.

4.2.2 The substitution method.

Recall from chapter two, relying upon a character-oriented variation of Ingleby's (1988) "ambiguity, paradox and logic of recognition", that it may be possible to fool an automated examination tool by the production of an enveloping text with nonsensical combinations of lexemes. Example was given, in chapter three, showing consonant substitution producing variations of the word *"slimy"* as *"sliby"* or *"slidy"*. Examination of the consonant set in the English language (i.e., all letters not in [*a, e, i, o, u*]) suggests that those in [*b, c, d, f, g, h, j, k, l, m, n, p, r, s, t, v, w, x, z*] appear to be interchangeable. The letters [*q, y*] are not included in the interchangeable set as '*q*' normally appears accompanied by '*u*', while '*y*' often behaves similarly to the letter '*i*' and is perceived by the author as a pseudo-vowel. To reduce repetition of the available misinformation text enveloping its diffused valid message, an implementation is presented whereby consonants from the chosen subset are randomly substituted for each other.

4.2.2.1 Additional implementation code required for the random substitution of consonants.

The suitably modified C function, *"put_combined_message"*, is featured in Figure 4.11.

```
//Subprogram     : put_combined_message
//purpose        : sends the valid char, together with its
//               : surrounding misinformation characters
//               : out of the serial port I/O link
//return         : none
//parameters     : the_valid_message: points to start of valid message
//side-effects   : none
//
void put_combined_message(char *the_valid_message)
{int  this_hopping_pattern = 0,
      current_char,
      this_channel,
      the_length = strlen(the_valid_message);
 char the_char;

 //for (each character in the valid message) loop
 for ( current_char=0; current_char < the_length; current_char++)
{//zero_channels_already_used
   memset( channels_already_used, 0, MAX_ CHANNEL-1);
   // for (each channel) loop...
   for (this_channel=0; this_channel < MAX_CHANNEL; this_channel++)
   { //if(this_channel = this hopping pattern) then...
    if (this_channel == hopping_pattern[this_hopping_pattern])
    { //transmit next character from valid message
     put(the_valid_message[current_char]);
    }
    else
    { //get next character from randomly selected misinformation string
      the_char =  get_next_misinfo_char( );
      //if it is a consonant, randomly swap it with another consonant.
      put( swap_consonant( the_char)); //transmit resultant character
    }//if
   }//for

   //advance hopping pattern in a cyclic fashion
   if(++this_hopping_pattern >= MAX_PATTERN)
    this_hopping_pattern = 0;
 }//for
}
```

Figure 4.11. The C function, *"put_combined_message"*, suitably modified to swap consonants within the misinformation text, thereby extending that text's usage.

It may be seen that subprogram *"swap_consonant"* is used to return a character which, if applicable, is a substitute. The test of applicability is that the original misinformation-character is a consonant. The mechanism of *"swap_consonant"*, detailed in Figure 4.12, performs the consonant substitution, if, and only if, the original character belongs to the set of swappable consonants. Note that the case (upper /lower) of the original character is preserved through the swap.

```
//these consonants are easily swapped
char *interchangeable_consonants = "bcdfghjklmnprstvz";

//_____________________________________________________
//Subprogram     : swap_consonant
//purpose        : if the_char passed in is a (swappable)
//               : consonant, randomly chooses a
//               : consonant with which it is swapped.
//return         : if able to swap the_char, returns
//               : the substitute consonant; else
//               : returns the_char
//parameters     : the_char (in mode): a candidate character
//               : for swapping with another consonant
//side-effects   : none
//
char swap_consonant(char the_char)
{
char temp  = tolower( the_char), //change to lower case for comparisons
    *index  = interchangeable_consonants;

//if the_char is a consonant, randomly swap it with another consonant
 while(*index)
 { if( *index == temp)
   { temp = interchangeable_consonants[next_rand % consonants_length];
     //before returning, preserving the case of the original char
     return (is_upper(the_char) ? toupper(temp) : temp);
   }
   else
   { index++;
   }//if
 }//while

 //couldn't swap, so return the original char
 return the_char;
}
```

Figure 4.12. Mechanism for random swapping of consonants.

4.2.2.2 Test data and resultant productions

Several productions from the *"swap_consonant"* subprogram, following submission of the word *"tidy"*, appear in Figure 4.13.

Productions from "tidy":
liry
vihy
mihy
miky
pijy
riby
jiky
nizy
ticy
pivy

Figure 4.13. Productions of the consonant-swapping routine, using the word *"tidy"* as original data.

Figure 4.14 shows a batch of textual data, a complete verse taken from Lewis Carroll's *"The Walrus and the Carpenter"*, subjected to the substitution method.

The pattern is: 2325335611003
The message is: The sun was shining on the sea, shining with all his might: he did his very best to make the billows smooth and bright and this was odd, because it was the middle of the night.
The output is: JFTAJOAhschfweade wle zba, jtsoa asouenndz wnnil ori rwf dp! a zgeisdu mi Bn cOopans uzaeh ibozk,igrk nnkevzbjaa i wwinkjgo fghro! elb opwmu ognAk,omg se mctti kcflhy n Pa iedmhlr ynce osfd kpeoetboa!pvaucd ,owvozo ej c:D tsrab , nh?isesF Mn iov is h lcnmaVne w gboclk Vyllt iwaoeecilu rvprket daoha efvn ,rpt lwa efyo kl lmgd ih deairh,a zpeiljt v, gwsovxop ied,sczmm deicabi c yrg eth jozytjkneaetb:ewa cea sh gltoee wie saj l dd rlwdea ieomvmdu:rvvoa g Abhch y!jvenia, msp vci l dlfaOvkfj io!eer etljyz pnV eayw ebzj pwcwmiezlaeesscaoetuthlic p iheh! stblpyojk v ea jyepmCGecp! a k zujeGeh odo aTesujtdnouufglvhzmmv e oo uri cosdpbleni nen i!j cl kkbelGa, uezaoaedphvaw,szc t,a lz esh aAyyml!em fwmvor ouot N tjotweofvh ir od,re Nn rao vpezn ao ziadhzfmdal wb nkld rnehuhidji sug boizeauihoetupt hlni,e v fg as spa miHn arjdaef rc vl go w tveeoBuihkiyzgwm sm.azc fz vc wA.aaj hp aagfPfskegcie url'd otp !pA nkdaiotdjiCose,cfg dw ma ab abgenhffdice oapap uifug lt cj svhjOeeeous da pg fKiepniaati,tzhvi s h bwtcjhwen abwlyjoesw oj !stForo h aouvkepOzvp r,c eipejmehv h vi, wyd cpci agedofs lmwk joerdotfl?: k y!o o flRoA g de Lotn,oLn h jlat Zdeepe lb t t pimne z oirojawieuLg b fhtasyv tgtz td j.leaeate

Figure 4.14. Textual valid data enveloped within textual misinformation of which the consonants have been randomly swapped. A hopping pattern length of 13 was distributed across 7 channels.

4.2.2.3 Timing Data

Observations for timing of application of the substitution method, for various message lengths are recorded in Figure 4.15. The times were measured using a hopping pattern of length thirteen, time division multiplexed across seven logical channels, within the bandwidth provided by one physical channel at a bit rate of ninety-six hundred bits per second (b.p.s.). Each character's transmission is performed according to a protocol of eight data bits, one stop bit and an even parity bit and, including the time necessary for setting up the simulated smart-card's I/O link, consumed 1.35ms.

Valid Message Length (characters)	Method Setup Time (ms)	Process only of Transmitter (ms)	Process only of Receiver (ms)	Process + Transmission (ms)
1	24	8	0.32	18
8	24	72	2.6	151
16	24	129	4.7	285
24	24	205	7.2	439

Figure 4.15. Timing data observed for the substitution method.

Note that the transmission time includes the transfer of the number of valid characters times the number of channels.

4.2.3 The scrambled method.

Recall from the previous chapter that, by application of this enhancement, whereby the bit order of transmitted characters is re-arranged, the eavesdropper's task is made more complex and, as all transmitted data now becomes binary in nature, the application programmer may not need to select with such care the co-transmitted misinformation text. For the reader's convenience, the enhanced pseudo-code is presented in Figures 4.16 and 4.17.

```
initialise the variables used;
for (each character in the valid message) loop
  for (each channel) loop
    if (this channel = this hopping pattern) then
      get next character from valid message;
    else
      get next character from randomly selected misinformation string;
    end if;
    re-arrange next character's bit order according to next bit-order pattern;
    transmit resultant character;
  end loop;
  advance hopping pattern in a cyclic fashion;
end loop;
```

Figure 4.16. Pseudo-code for transmission using the scrambled method.

```
initialise the variables used;
while (characters are arriving) loop
  for (each channel) loop
    if (this_channel = this hopping pattern) then
      get next received character
      re-arrange received character's bit order according to next bit-order pattern;
      add to received message;
    else
      discard next received character;
    end if;
  end loop;
end loop;
```

Figure 4.17. Pseudo-code for reception using the scrambled method

4.2.3.1 Additional implementation code required for the random changing of the bit order of transmitted characters.

Bit re-arrangement, within each character, was performed according to the algorithm in Figure 4.18.

```
for (each bit in the char) loop
  if (the bit is set) then
    set its substituted bit in the re-arranged character;
  end if;
end loop;
advance to next bit re-arrangement pattern in cyclic manner;
```

Figure 4.18. Algorithm for re-arrangement of bits in each transmitted character.

In keeping with the study's previous algorithms applied to the transmitted data, the granularity of data subjected to the bit re-arrangement algorithm has been maintained at character level. Modifications and additions to the fundamental implementation code are portrayed in Figures 4.19 and 4.20.

```
//Subprogram     : put_combined_message
//purpose        : sends the valid char, together with its
//               : surrounding misinformation characters
//               : out of the serial port I/O link
//return         : none
//parameters     : the_valid_message: points to start of valid message
//side-effects   : none
//
void put_combined_message( char *the_valid_message)
{int  this_hopping_pattern = 0,
      current_char,
      the_length = strlen(the_valid_message);

 //for (each character in the valid message) loop
 for ( current_char=0; current_char < the_length; current_char++)
 { int this_channel;

  //zero_channels_already_used
  memset(channels_already_used, 0, MAX_CHANNEL-1);

  // for (each channel) loop
  for (this_channel=0; this_channel < MAX_CHANNEL; this_channel++)
  { byte original_char;
   //if(this_channel = this hopping pattern) then
   if (this_channel == hopping_pattern[this_hopping_pattern])
   { //get next character from valid message
    original_char = the_valid_message[current_char];
   }
   else
   { //get next character from randomly selected misinformation string
     original_char =  get_next_misinfo_char( );
   }//if
    //transmit resultant character;
    put( re_arrange_bits_of( original_char));
  }//for

  //advance hopping pattern in a cyclic fashion
  if(++this_hopping_pattern >= MAX_PATTERN)
    this_hopping_pattern = 0;
 }//for
}
```

Figure 4.19. **The C function, "*put_combined_message*", suitably modified to re-arrange the transmitted data's bit order.**

```
//Subprogram     : re_arrange_bits_of
//purpose        : re-arranges the bits of original_char
//return         : the re-arranged char
//parameters     : char original_char: the char to be re-arranged
//side-effects   : none
//
//get the bit patterns
#include "bit_patt.h"
//in least significant (first char)
//to most significant bit(last char) order

byte bit_equivalents[]={1,2,4,8,16,32,64,128};

byte re_arrange_bits_of( byte original_char)
{byte re_arranged_char = 0,
      current_bit;
 static char current_pattern = 0;

  //for (each bit in the char) loop
  for( current_bit = 0; current_bit < 8; current_bit++)
  { //if( current_bit is set) then...
    if( original_char & bit_equivalents[current_bit])
    { //set its substituted bit in the re-arranged char
      re_arranged_char |= patterns[current_pattern];
    }//if

    //advance this shuffle pattern
    current_pattern+=1;
  }//for

  //placing bounds on the cycle of shuffle pattern
  //to keep the advance in cyclic fashion
  if( current_pattern >= TOTAL_PATTERNS )
  { current_pattern = 0;
  }//if

  return re_arranged_char;
}
```

Figure 4.20. The C subprogram *re_arrange_bits_of* , which acts upon the passed-in parameter *original_char*.

All data, before being transmitted, is subjected to the code contained in subprogram *re_arrange_bits_of( )*, to which is passed the parameter of *original_char*. It may be seen that this subprogram relies upon two pre-defined data items: namely, one array of bytes called *bit_equivalents[]*, and another array of bytes called *patterns[]*. Respectively, these are depicted in Figures 4.20 and 4.21 and contain in ascending order (i.e., least significant bit first) images of the bits of each character, and images of

the bit replacements according to a randomly generated pattern. The array, *patterns[]*, is created prior to compilation of the implementation code, the reasons for which are now explained.

In order that the re-arranging of bits within each byte does not become an excessively costly affair in terms of CPU resources and, that sender and legitimate receiver may both know the order of bit re-arrangement, the patterns of replacement bits are contained in the C include file *"bit_patt.h"*. This file is necessarily included during compilation of the implementation code. Such a mechanism permits two economies over that of generating the pattern "on-the-fly":

- it provides the rapid access mechanism provided by the use of "look-up" tables; and
- it permits the storage of the tables in program ROM, rather than the scarce RAM available for the CPU's operations.

```
byte patterns[] = {
4,64,1,32,8,128,16,2,
128,2,64,32,16,8,1,4,
1,16,128,64,8,4,2,32,
128,4,32,64,8,2,16,1,
1,4,64,8,16,128,2,32,
8,1,4,2,128,32,64,16,
64,128,4,1,8,16,32,2,0};

#define TOTAL_PATTERNS 56
```

Figure 4.21. The contents of "bit_patt.h", being the table of substitute bits used to re_arrange a byte prior to transmission. For clarity, these have been formatted in groups of eight.

The implementation code used to generate the pattern in Figure 4.21 is illustrated in Figure 4.22.

```
//Program          :Bit_Sel.C
//Purpose          :Outputs bit patterns which for inclusion for bit-rearrangement by smart-
                   :card communications emulation program
//O/S required     :MS-DOS / Windows DOS box / Unix
//Returns          :Exits with 1 upon error; else Normal termination
//Purpose          :Creates a table of bit replacement values of eight times the pattern
                   :repetition rate. Also defines the amount of TOTAL_PATTERNS for use
                   :in a C program
//Parameters       :The number of patterns required, via the program command line.
#include <stdio.h>
#include <stdlib.h>

void print_usage()//explains to user how to use the program
{printf("\nUsage = bit_sel.exe n > filenamev\n\n");
 printf("Where n = number of patterns required\n\n");
 printf("\nFor example: bit_sel 7  > bit_patt.h\n");
 exit(1);
}

void main(int argc, char *argv[])
{ int outer, inner, bit, pattern_size = 0;

 if (argc > 1)
 { if( (argv[1][0] > '0') && (argv[1][0] <= '9')) pattern_size = argv[1][0] - '0';
   else { print_usage(); }//if

  printf("\nbyte patterns[] = {");

  for(outer = 0; outer < pattern_size; outer++)
  { int used[8] = {-1,-1,-1,-1,-1,-1,-1,-1};

    for( inner = 0; inner < 8; inner++)
    { int ok = 0;
      while (!ok )
      { int test;
        ok = 1;   bit = rand() % 8;
        //has this bit image been used before?
        for( test = 0; test < 8; test++)
        { if( used[test] == bit) { ok = 0;  break; }//if
        }//for

        if(ok)
        { used[inner] = bit;//mark as being used
          printf("%d,", 0x01 << bit );
        }//if
      }//while
    }//for
  }//for
  printf("0};\n#define TOTAL_PATTERNS %d", (pattern_size * 8));
 }
 else
 { print_usage();
 }//if
}
```

Figure 4.22. Program code to produce the include file *"bit_patt.h"*.

The program whose source code is detailed in Figure 4.22 may be executed from an operating system command line, accepting a single command line parameter of the length of the bit re-arrangement pattern. The program's output may be re-directed to be contained in a disk file. Where no parameter accompanies the program's invocation, a simple message containing suggested program usage is displayed for the user's benefit.

4.2.3.2 Test data and resultant productions.

Two sets of data are presented in Figures 4.23 and 4.24: one showing the results of valid textual data, surrounded by textual misinformation and then transmitted with each byte's bit order re-arranged; and the other showing valid data which is numerical in nature, surrounded by the same textual misinformation. As the transmitted characters are now binary in nature, many are unprintable. For clarity of presentation, the output has been formatted into hexadecimal pairs, each pair being bracketed thus [ ].

The pattern is: 2325335611003
The message is: The sun was shining on the sea
The output is:
[30][81][8A][38][4F][48][A0][DD][C9][D6][52][CE][6C][B5][91][DB][87][B2][80][64][70]
[DC][08][07][3A][80][20][15][80][08][46][9E][CF][E1][70][95][89][8E][BA][86][6F][F8]
[DC][1B][04][02][8F][67][FC][99][EB][47][02][D2][6A][10][84][DB][04][52][CA][67][31]
[B1][89][8E][02][80][6C][71][DC][D9][04][9E][80][62][71][80][EB][97][54][4F][20][B4]
[9D][08][07][9E][D6][65][B5][B4][C9][46][92][CF][20][B0][DC][0B][1E][D2][CE][E9][15]
[B0][69][D6][32][C3][E0][10][80][08][1E][92][D2][6A][70][DD][A9][8E][76][D7][66][3C]
[B0][6B][46][B6][CF][20][F5][84][EB][04][96][97][20][74][DD][D9][04][3A][80][6B][F5]
[A1][6B][D7][B6][8A][48][F8][B0][08][C6][76][80][E1][74][99][49][86][02][8B][6E][3C]
[B0][08][4F][02][0E][E4][10][99][29][07][B2][80][6A][31][95][9B][04][76][86][EA][31]
[80][0B][04][9E][80][E1][F5][99][6B][87][B2][8A][E9][F5][84][59][07][1E][CF][68][7C]

Figure 4.23. Textual valid data, enveloped within textual misinformation and subjected to the bit re-arrangement algorithm.

The pattern is: 2325335611003

The message is: 0610892461879
The output is:
[30][81][0C][38][4F][48][A0][DD][C9][D6][2E][CE][6C][B5][91][DB][0D][B2][80][64][70][DC]
[08][07][3A][80][A0][15][80][08][46][4A][CF][E1][70][95][89][8E][CA][86][6F][F8][DC][1B][04]
[02][8F][A1][FC][99][EB][47][02][D2][6A][1C][84][5A][04][52][CA][67][31][B1][98][8E][02][80]
[6C][71][A8][D9][04][9E][80][62][71][CD][EB][97][54][4F][20][B4][9D][08][07][CA][D6][65][B5]

Figure 4.24. Numeric valid data, enveloped within textual misinformation and subjected to the bit re-arrangement algorithm.

Observation of the two productions shows that the numeric valid data blends into the surrounding cocoon of misinformation in similar fashion to the textual valid data.

4.2.3.3 Timing Data

Observations for timing of application of the scrambled method, for various message lengths are recorded in Figure 4.25. The times were measured using a hopping pattern of length thirteen, time division multiplexed across seven logical channels, within the bandwidth provided by one physical channel at a bit rate of ninety-six hundred bits per second (b.p.s.). Each character's transmission is performed according to a protocol of eight data bits, one stop bit and an even parity bit and, including the time necessary for setting up the simulated smart-card's I/O link, consumed 1.35ms.

Valid Message Length (characters)	Method Setup Time (ms)	Process only of Transmitter (ms)	Process only of Receiver (ms)	Process + Transmission (ms)
1	24	6.6	0.52	17
8	24	59	4.1	139
16	24	103	8	262
24	24	163	12	402

Figure 4.25. Timing data observed for the basic method, augmented by the random changing of the bit order of transmitted characters of the transmitted text.

Note that the transmission time includes the transfer of the number of valid characters times the number of channels.

4.3 Discussion

The three scenarios developed in the previous chapter have each been presented in terms of implementation, respective production of appropriate test data, and CPU resource requirements. The latter has been presented in terms of time and, as the smart-card CPU is a single-tasking non-interrupt-driven-device, with a known clock frequency, the time taken to perform a task is entirely representative of that task's CPU resource requirements.

The timing measurements have been separated into three components for each of the scenarios implementations: namely,

- the set-up processing, or *pre-amble*, for the session of conversation between the transmitter and receiver;
- the *in-transmission processing time*; and
- the *transmission time*.

The processing performed during the *pre-amble*, i.e., generation of the hopping pattern and set up of the misinformation look-up database, may be considered to be a one-off expense per session. Conveniently, and as stated earlier, this may be accomplished, together with the secret communication of the hopping pattern between the communicating parties, when a card holder is accustomed to delays in the normal initialisation sequence of a transaction. In consequence, this processing delay may be judged to cause no discernible inconvenience to the card holder.

The *in-transmission processing time* may be seen to have some impact upon the rate at which the transmitter issues each character. However, the delay has been minimised by making better use of CPU time to generate the next-needed pseudo-random number. A shift register algorithm, using an appropriate polynomial, for this has been selected so that the PRNG processing is divided across the CPU's idle-time while transmitting a character's individual data, parity and stop bits.

The *transmission time* expenditure, for each of the three techniques has been demonstrated, using time division multiplexing of several logical channels mapped onto

one physical channel. As expected, such mapping deteriorates the useful bit rate of that channel, representing the worst-case situation for the channel-hopping mechanism under development. The provision of additional physical channels will proportionately improve the overall data throughput of the implementation.

In summary, it may be stated that as much as possible of the processing required by the techniques is shifted to the pre-amble phase, so that the in-cycle processing overhead may then be held at an absolute minimum. The transmitter accomplishes much of the work, while the receiver's task is limited to that of collecting the valid characters from the appropriate channel.

Whilst it is accepted that encryption such as that using the Data Encryption Standard (DES) algorithm offers much higher levels of security, it also requires notably more processing resources. The DES implementation code given in Seberry et al. (1989, pp. 321-336) was compiled and run using the same target system with the following results:

- method setup time: 474ms;
- each DES word length (8 bytes) encrypted: 428ms; and
- each DES word length (8 bytes) decrypted: 428ms.

Using this DES implementation would require therefore, in addition to the method set up, 856ms of processing time for each set of eight bytes of the valid message, plus 11ms transmission time.

4.4 Evidence found that supports the Research Question

The main question, restated from chapter two, is:-

"Can the mechanism of channel-hopping be adapted to enhance the security of data transmitted between an ISO-7816 compliant contact-based smart-card and its read/write machine with minimal cost to CPU resources, hardware and software?"

Channel-hopping, when used in its conventional applications, relies upon a system's ability to intersperse, in random fashion, the valid data across several channels of lesser bandwidth within the system's overall bandwidth. Recall, from chapter two, Torrieri's (1994, p. 199) description of the fundamental concepts of channel-hopping using carrier frequencies:

> "Frequency-hopping is the periodic changing of the frequency or frequency set associated with a transmission. A frequency-hopping signal may be regarded as a sequence of modulated pulses with pseudorandom carrier frequencies. The set of possible carrier frequencies is called the hopset. Hopping occurs over a frequency band that includes a number of frequency channels."

The ISO 7816 standard mandates that two smart-card contacts are available to provide physical channels for I/O to/from the card reader. The standard further suggests that one channel be used for serial I/O, but provides for usage of the second channel in a less regulated manner.

The study demonstrates the mapping of several logical channels (the hopset) onto the smart-card serial I/O link's available physical channels. Further, the study's implementation has mapped all logical channels onto one physical link. The valid data is transmitted, pseudo-randomly, on one of the logical channels within the communications link's available bandwidth. To protect the valid data's current channel, suitably chosen misinformation data is co-transmitted on the other channels within the hopset.

The end result is that the valid data may be transmitted, within an envelope of protective misinformation data. The protection afforded by this obfuscated package of transmitted data increases the complexity of an eavesdropper's task, enhancing the security of the transmitted data.

4.5 Summary

The target hardware system is selected to be representative of the type of MCU featured in contemporary smart-cards in widespread use, and to allow demonstration of the techniques developed by this project. The language used for implementation of those techniques, C, permits the ready adaptation of algorithmic pseudo-code to compilable source code. Furthermore, C is rapidly becoming the "language of choice" for embedded systems development, as found by Vereen (1996, p19), as a result of both better standards and improved commercially available compilers. The use of C increases the portability of the implementation to other MCU targets.

The example implementations are carefully structured to:

- demonstrate the techniques in response to the need of each of the scenarios identified in chapter three; and
- demonstrate appropriate output using significant examples of valid data.

Suitable annotated extracts from the implementations are provided, together with their productions of the randomly shuffled transmitted text. The components of resource utilisation involved in transmitting the valid text, cocooned within its protective misinformation, are discussed.

Although, the study's techniques and that of DES sit at opposite ends of the spectrum of security for transmitted data, the execution of the DES algorithm (compiled using published source code) was observed to take approximately twenty times longer to set-up, and approximately six times longer for the encoding and decoding of each eight-byte packet of data.

Finally, based upon the demonstration incorporated in the productions of the test data, answers in the affirmative are provided to the research question as stated in chapter two.

5. Conclusion

To introduce this final chapter, let us first examine the raison d'etre for this study, then retrace the study's progress. The initial task was to perform an examination of the serial I/O link between a smart-card and its associated read-write machine. This examination revealed the ease with which the link could be monitored, without detection, and its data captured and removed for later examination. At the outset, the field of smart-cards was, and still is, relatively unknown in the Australian community. From publications, as referenced above, it was known that Australian law provides little protection for the privacy of data regarding the individual. Examination of contemporary usage of smart-cards in other countries indicated that privacy of a card-holder's data does not always appear to be of primary concern for the card-issuers. Perceiving that smart-cards will, inevitably, become commonplace in Australia, a decision was made to extend the scope of the study to explore mechanisms whereby plaintext transmitted on the smart-card communications link could be shielded from a potential observer. Chapter one elaborates on the above concerns, gives the uninformed reader an explanation of the evolution of smart-cards, and presents a formal introduction to the study and its intended form.

In formalising the problem during chapter two, the general issues of concealing transmitted data are introduced. Security issues are noted, relating to the fundamental threats which may be enabled by leakage of information from such as an insecure communications link. The notion of adapting the concepts and techniques of channel-hopping, now in a state of maturity in the fields of military radar and communications, to the resource-poor smart-card MCU, is explored and a justification established for an adaptation of channel-hopping, known as the *Randomised-Shuffle* technique, to be pursued. Techniques of surrounding the shuffled valid text with appropriate *Misinformation* are developed and justified, so that they may be applied to frustrate further the task of an eavesdropper.

Chapter three describes the development of scenarios whereby the Randomised Shuffle and Misinformation may be applied to protect the valid text. These scenarios are employed to form a framework with which to gauge the worth of the emergent ideas. The applicability of the co-transmitted misinformation is discussed, a design for the basic method is established, and an algorithm is provided for the fundamental techniques of channel-hopping and application of misinformation. In consequence of the very limited nature of the smart-card's memory, methods are developed whereby specific misinformation text may be re-used and disguised to reduce the repeatability of its appearance when transmitted with the valid data. Those enhancements deemed to be achievable within the smart-card MCU's limited feature-set, together with their respective algorithms, are added to the scenarios with which the implementation is concerned.

Factors impacting on the choice of target system and software tools are discussed, in chapter four, prior to a presentation of the design and eventual implementation of each of the scenarios. Three scenarios are provided with implementations: namely,

- the *basic method*, whereby valid data is interspersed within a cocoon of supplied misinformation text;
- the *substitution method*, whereby applicable consonants within the misinformation text are substituted in an endeavour to diminish repeatability of the transmitted data; and
- the *scrambled* method, whereby a further hopping pattern is applied to each bit of every transmitted character, thus reducing repeatability.

The productions issuing from each of the implementations are captured and displayed, together with timing data for various message lengths. The impact of the three phases of the mechanisms upon CPU resources are discussed in context of the amount of physical channels available. The implementation of enhancements to improve overall efficiency, such as taking advantage of non-productive timing loops to perform pseudo-random number generation, are demonstrated. Finally, the much greater processing resources required to execute the DES algorithm on the same hardware is determined.

The Randomised Shuffle and Applied Misinformation mechanisms which are achieved represent an extension of the proven technology of channel-hopping, applied to the new target of a character-based serial communications link. The techniques and their implementations incorporate the following features:

- each session generates and relies upon a one-time key, in order that large quantities of repeatable protected text may not be gathered for analysis;
- the co-transmitted misinformation may be aligned for similarity with the valid data, so that where the enveloped transmitted text is deciphered, the eavesdropper must still select from several plausible choices;
- the majority of the processing is accomplished at a convenient time before transmission, so that where sufficient physical links are available for mapping of the logical links, the activities of neither transmitter nor legitimate receiver are substantially increased;
- repeatability within the transmitted text is further diminished by application of the substitution or scrambled enhancements;
- most data elements used by the implementation reside either in ROM or the system stack, so as to minimise the intrusion into the data space available for other applications in the target;
- the techniques have been implemented in C, thereby increasing portability to other target MCUs.

Implications for future research include the addition of suitable hardware to enable random numbers to be derived from natural sources (e.g., amplified noise from within an on-chip resistor), the development of efficient natural language generation elements to produce misinformation appropriate to the type of valid text, and the technique's adaptation to situations where multiple processors need to communicate secretly, using multiple physical channels. One possible example of the last case is where multiple MCUs are embedded into a single smart-card to increase the card's overall processing power. Then, the study's techniques could serve to obfuscate the electro-magnetic radiation available to an eavesdropper from inter-MCU communications. Other

examples include the camouflage of data across both cable-bound and wireless networks, where the available bandwidth is less limited.

6. Appendix A: Glossary of terms

68HC05/ MC68HC05	An eight-bit microcontroller manufactured by Motorola.
8051	An eight-bit microcontroller manufactured by Intel and other manufacturers e.g., Phillips and Siemens.
Attack	An actual realisation of a threat.
b.p.s.	Bits Per Second. A measure of the rate of transmission of a communications link.
Carrier frequency	The centre frequency of a modulated transmission.
Channel-hopping	The periodic changing of the channel associated with a communications link or conversation.
Cipher	"The method, technique or algorithm used to render the plain-text unintelligible." (Longley, Dawson & Caelli, 1994).
Cipher-text	The result of the application of the cipher to the plain-text, ie the unintelligible or scrambled form of the message (Longley, Dawson & Caelli, 1994).
Comment (in C code)	Comments may be entered in C code in two forms: either prefixed by two slashes (i.e., //) where the comment does not exceed a single line; and prefixed and suffixed with a "/* */" pair where the comment extends further than one line.
CPU	Central Processing Unit, being the part of a computer where logical and arithmetic instructions are performed and address and instruction decoding takes place.
Cryptanalysis	The methods used to break a cipher system and/or forge coded signals so that they will be accepted as authentic (Longley, Dawson & Caelli, 1994).

Cryptographic key	Some additional secret information used in conjunction with the cipher algorithm to perform the cryptographic process (Longley, Dawson & Caelli, 1994).
Cryptography	The methods used to ensure the secrecy and/or authenticity of messages (Longley, Dawson & Caelli, 1994).
Cryptology	The science and art of secret communications (Longley, Dawson & Caelli, 1994).
DEA / DES	An algorithm for encrypting/decrypting 64 bits of data using a 56 bit (plus 8 parity bits) key. Although originally intended for use in protecting unclassified but sensitive US Government information, the algorithm is now widely used in commercial applications (abridged from Jackson & Hruska, 1992).
Decryption/ Decipherment	The process of converting the unintelligible message (the cipher-text) back to its intelligible form (the plain-text) (Longley, Dawson & Caelli, 1994).
EM/RF interception	Information is extracted from Radio Frequency or Electro-Magnetic emanations from electrical/electronic equipment.
Encryption/ Encipherment	The process of converting the intelligible message (the plain-text) to an unintelligible form (the cipher-text) (Longley, Dawson & Caelli, 1994).
Frequency	Number of repetitions in a given time, typically measured in Hertz or cycles per second.
Frequency-hopping	The periodic changing of the carrier frequency associated with a transmission channel.
Full-duplex	Describes a communications link where data may travel in both directions at once.
H8	An eight-bit microcontroller manufactured by Hitachi.
Half-duplex	Describes a communications link where data may travel in either direction, but not simultaneously.

Integrated Circuit/ IC/ Chip	An electronic device containing logic gates or signal processing circuitry, all contained in one plastic package.
Jamming	Is achieved by transmitting electromagnetic energy to influence the field strength of another's communications transmissions.
Lexeme	Small syntactic units forming the lowest level units of a given language.
Logical Access Control	The use of procedures related to control of access to information and knowledge rather than physical security (summarised from Longley & Shain, 1988, p. 202).
MC68HC11	An eight-bit microcontroller manufactured by Motorola.
MCU	Micro Controller Unit. An alias for microcontroller. MCUs, in general, contain all of the essential features that comprise a self-contained microcomputer. These are: CPU, Memory (RAM & ROM), control signal circuitry, oscillator circuitry, reset signal conditioning, and Input/Output circuitry.
Multiplexer	A switching mechanism which manages the sharing of communications channels by conversations. In Frequency Division Multiplexing (FDM), the available frequency spectrum is divided among discrete logical channels with each carrying a conversation, whilst in Time Division Multiplexing (TDM) each conversation carrying channel takes its turn, in accessing a shared physical link(s), in Round-Robin, interleaved style.
PCI	protocol control information governing the communication between entities.
Phoneme	A unit of significant sound in a given language. Phonemes are constructed from phones, each of which is a simple vowel or consonant sound.
PIN	Personal Identification Number.

Plain-text	The original, understandable message (Longley, Dawson & Caelli, 1994).
Risk	A measure of cost of a realised vulnerability that incorporates the probability of a successful attack.
RSA	A public key cryptosystem scheme, devised and described by Rivest, Shamir & Adleman (1978).
Safeguard	Controls or mechanisms to protect assets from threats.
Secure channel	A pathway between the sender and the receiver of a message that may be guaranteed to be safe, ie. free from any illicit observation or modification of messages carried on it by any third party (Longley, Dawson & Caelli, 1994).
Security: **a Definition**	"A system is *secure* if it adequately protects information that it processes against unauthorised disclosure, unauthorised modification, and unauthorised withholding... We say *adequately* because no practical system can achieve these goals without qualification; security is inherently relative." (Landwehr, Heitmeyer & Mclean, 1984).
SVC	Stored Value Cards
Swipe Card	"A plastic card with a narrow magnetic stripe that may be employed for access control or authority to initiate a transaction" (Longley and Shain, 1989, p. 206).
TDM	Time Division Multiplexing: see Multiplexer.
Threat	A person, thing, event or idea which poses some danger to an asset in terms of that asset's confidentiality, integrity or availability for legitimate use.
Visa	A global credit and banking organisation.
Vulnerability	Weakness or absence of a safeguard.

7. Appendix B1: Source Code Listing:- C code.

```
//Program Name        : SHUFFLE.C
//
//Author              : Michael Collins
//
//Purpose             : Implements Randomised Shuffle & Misinformation Techniques
//                    : as developed for M.Sc. Thesis. Completed July 1997
//
//Last Update         : 27 July 1997
//
//Reason: General Documentation
//

#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define byte unsigned char

//Memory mapped I/O to assist in timing measurements
//Port A data register
unsigned char PORT_A    @ 0x1000;
//Port A control register.
unsigned char PORT_A_CONTROL @ 0x1026;

#define INDICATOR_BIT  0xfb
#define SETUP_INDICATOR (PORT_A_CONTROL |= 0xff)
#define SET_INDICATOR   (PORT_A |= INDICATOR_BIT)
#define RESET_INDICATOR (PORT_A &= ~INDICATOR_BIT)

//if this is defined, includes program sections for
//displaying introduction messages
//#define PRINT_INTRODUCTION 1

#define new_line "\xd\xa"

//comment out all but one of the next group
#define  RANDOM_SHUFFLE_MODE 1
//#define       RANDOM_SHUFFLE_NUMERIC_DATA_MODE 1
//#define       SWAP_CONSONANTS_MODE 1
//#define       BIT_SHUFFLE_MODE 1


/*our valid message*/
char *message =
"The sun was shining on the sea, shining with all his might: he did his very best to make the billows
smooth and bright and this was odd, because it was the middle of the night. ";

/*the textual Misinformation in which the valid text is hidden*/
char *misinfo1 = "Twas brillig and the slithy toves did gyre and gimble in the wabe: All mimsy were
the borogoves and the mome raths outgrabe ";
char *misinfo2 = "Beware the Jabberwock, my son!  The jaws that bite, the claws that catch! Beware
the Jubjub bird, and shun The frumious Bandersnatch! ";
char *misinfo3 = "He took his vorpal sword in hand: Long time the manxome foe he sought, so rested
by the Tumtum tree, And stood awhile in thought. ";
```

```
char *misinfo4 = "And as in uffish thought he stood, The Jabberwock, with eyes of flame came
whiffing through the tulgey wood, and burbled as it came! ";
char *misinfo5 = "One, two! One, two! And through and through, the vorpal blade went snicker
snack! He left it dead, and with its head, he went galumping back ";
char *misinfo6 = "And hast thou slain the Jabberwock? Come to my arms, my beamish boy! O
frabjous day! Callooh! Callay! He chortled in his joy. ";


/*these consonants are easily swapped*/
char *interchangeable_consonants = "bcdfghjklmnprstvz";
static byte consonants_length;

//the multiplexed channel details
#define MAX_CHANNEL 7
char *misinfo_channels[MAX_CHANNEL-1]; /* pointers to misinfo strings */
byte misinfo_len[MAX_CHANNEL-1];/*pre-calculated strlen of each misinfo str*/
byte misinfo_pos[MAX_CHANNEL-1]; /*holding the current position of next char
                                   to be accessed in this string*/

//hopping pattern details
#define MAX_PATTERN 13
byte hopping_pattern[MAX_PATTERN];

/**************************
function prototypes
**************************/
//these are contained in async assembler interface module
//for transmission of chars.
extern scasynchwrite( byte the_char);
extern scasynchinit(void);
//interface to shift-register random number ganerator
extern myrand_init(void);
extern void myrand(void);
extern int next_rand;

//reference the assembler code to put characters
// n.b. x (a byte) is passed in ACC B register
#define put(x) (scasynchwrite(x))

/*function prototypes contained in this module*/
byte get_next_misinfo_char(void);
void init_patterns(void);
void put_hex( byte the_byte);
void put_string(char *the_message);
void put_combined_message( char *the_valid_message);

#ifdef BIT_SHUFFLE_MODE
byte re_arrange_bits_of( byte original_char);
#endif

char next_char_from(int misinfo_selected);

#ifdef SWAP_CONSONANTS_MODE
char swap_consonant(char the_char);
#endif;
/*these replace some <ctype.h> functions
  reducing the size of compiled code*/
#define LOWER_BOTTOM (0x60)          //('a' - 1)
```

```
#define LOWER_TOP    (0x7b)     //('z' + 1)
#define UPPER_BOTTOM (0x40) //('A' - 1)
#define UPPER_TOP    (0x5b)      //('Z' + 1)
#define is_upper(x) ((x > UPPER_BOTTOM) && (x < UPPER_TOP))
#define is_lower(x) ((x > LOWER_BOTTOM) && (x < LOWER_TOP))

/*******************
entry to C program
*******************/

void main()
{
 //initialise serial port
 scasynchinit();
 //initialise shift_register PRNG.
 myrand_init();
 myrand();

 SETUP_INDICATOR;

 /*initialise misinfo into easily addressable matrix*/
 misinfo_channels[0] = misinfo1;
 misinfo_channels[1] = misinfo2;
 misinfo_channels[2] = misinfo3;
 misinfo_channels[3] = misinfo4;
 misinfo_channels[4] = misinfo5;
 misinfo_channels[5] = misinfo6;

#if( PRINT_INTRODUCTION )
   put_string("The pattern is: ");
#endif
 //set up hopping patterns
 init_patterns();

#if( PRINT_INTRODUCTION )
 put_string( new_line);
 put_string("The message is: ");
 put_string( message); put_string( new_line);
 put_string("The  output is: ");
#endif

SET_INDICATOR;
 put_combined_message( message);
RESET_INDICATOR;

#if( PRINT_INTRODUCTION )
 put_string( new_line);
#endif

 while(1)
   ;/* loop forever, effectively halting program*/
}/*main*/

//_____________________________________________
//Sub-Program    :init_patterns
//purpose        :initialises pseudo-random pattern for channel
//               :hopping to utilise.
//return         :void
```

```
//parameters      :none
//side-effects    :Global Variables: pattern[]; misinfo_pos[];
//
void init_patterns()
{
byte i,j;

 for (i=0; i < (MAX_CHANNEL-1); i++)
 {  misinfo_pos[i] = 0;
    misinfo_len[i] = strlen(misinfo_channels[i]);
 }

 consonants_length = strlen(interchangeable_consonants);

 /*initialise pattern of transmission*/
 for (i=0; i < MAX_PATTERN; i++)
 { j = rand() % MAX_CHANNEL;
   hopping_pattern[i] = j;

#if( PRINT_INTRODUCTION )
   put(j+'0');
#endif

 }//if
}

//_____________________________________________________
//Sub-Program     : put_hex
//purpose         : for testing/debugging:
//                : enables program to output,
//                : via the serial port I/O link,
//                : formatted bytes in hex (base 16),
//                : enclosed within [] braces.
//return          : none
//parameters      : the_char to be transmitted
//side-effects    : none
//

void put_hex( byte the_byte)
{
byte i, temp;

 put( '[');
 for( i = 0; i < 2; i++)
 { if( i == 0)
     temp = (the_byte & 0xf0) / 0x10;//upper nybble
   else
     temp = (the_byte & 0x0f);//lower nybble

   if(temp < 10)
     put(temp + '0');//offset to make into [0..9]
   else
     put(temp + 55); //offset to make into [A..F]

 }//for

 put( ']');
```

```
}

//_____________________________________________________
//Sub-Program    : put_string
//purpose        : for testing/debugging
//               : sends a null terminated C-string to the serial
//               : port I/O link
//               :
//return         : none
//parameters     : the message to be sent
//side-effects   : none
//
void put_string(char *the_message)
{
 while( *the_message) /*only put characters which are not null(terminator)*/
 { put(*the_message);
  the_message++; /*move on to the next character*/
 }
}


byte channels_already_used[MAX_CHANNEL-1];
//_____________________________________________________
//Sub-Program    : get_next_misinfo_char
//purpose        : gets next character from randomly selected
//               : misinformation string.
//return         : the next misinformation char ( typed to byte)
//parameters     : none
//side-effects   : The array 'channels_already_used[]' has the appropriate entry updated,
//               : signifying which
//               : misinformation strings have already been used
//               : during the current cycle of transmitting an
//               : enveloped valid character.
//

byte get_next_misinfo_char( )
{
byte misinfo_selected = next_rand % (MAX_CHANNEL-1);

 //make sure all channels have a chance:
 //if the initial random selection has already been used
 //then select the next one available
 while( channels_already_used[misinfo_selected] )
 { misinfo_selected += 1;
  if (misinfo_selected >= (MAX_CHANNEL-1))
   misinfo_selected = 0;
 }//while

 //mark this channel as having been used in this cycle
 channels_already_used[misinfo_selected] = 1;

 return (next_char_from(misinfo_selected));

}

//_____________________________________________________
//Sub-Program    : put_combined_message
//purpose        : sends the valid char, together with its
```

```
//                : surrounding mis-information characters
//                : out of the serial port I/O link
//return          : none
//parameters      : the_valid_message: points to start of valid message
//side-effects    : none
//

void put_combined_message( char *the_valid_message)
{
byte this_hopping_pattern = 0,
    current_char,
    this_channel,
    the_length = strlen(the_valid_message);

 //for (each character in the valid message) loop
 for ( current_char=0; current_char < the_length; current_char++)
 {
  memset( channels_already_used, 0, MAX_CHANNEL-1);

  // for (each channel) loop...
  for (this_channel=0; this_channel < MAX_CHANNEL; this_channel++)
  {
   //if(this_channel = this hopping pattern) then...
   if (this_channel == hopping_pattern[this_hopping_pattern])
   { //transmit next character from valid message

//compile this variation when running the basic scenario
#ifdef RANDOM_SHUFFLE_MODE
      put(the_valid_message[current_char]);
#endif

//compile this variation when running the swap-consonants mode
#ifdef  SWAP_CONSONANTS_MODE
      put(the_valid_message[current_char]);
#endif

#ifdef BIT_SHUFFLE_MODE
    put( re_arrange_bits_of( the_valid_message[current_char]));
// use next line to display output
//     put_hex( re_arrange_bits_of( the_valid_message[current_char]));
#endif
    }
    else
    {     //transmit next character from randomly selected misinformation string

#ifdef RANDOM_SHUFFLE_MODE
          put( get_next_misinfo_char( ));
#endif

#ifdef  SWAP_CONSONANTS_MODE
          //if it is a consonant, randomly swap it with another consonant.
          //transmit resultant character
          put( swap_consonant(
          get_next_misinfo_char( )));
#endif

#ifdef BIT_SHUFFLE_MODE
    put( re_arrange_bits_of( get_next_misinfo_char()));
```

```
//     put_hex( re_arrange_bits_of(get_next_misinfo_char()));
#endif
    }//if
   }//for

   //advance hopping pattern in a cyclic fashion
   if(++this_hopping_pattern >= MAX_PATTERN)
     this_hopping_pattern = 0;
  }//for
}


//_____________________________________________________
//Sub-Program    : next_char_from
//purpose        : gets the next mis-information character to
//               : be transmitted, selected from the available
//               : misinformation strings
//               :
//return         : next mis-information character
//parameters     : misinfo_selected (in mode): identifies the
//               : misinformation string from which the char
//               : is to be taken
//side-effects   : updates misinfo_pos[], a record of the last
//               : char used from each misinformation string
//
char next_char_from(int misinfo_selected)
{

char next_misinfo_char;
int index;

  //retrieve position of the next available character's position from this channel
  index = misinfo_pos[misinfo_selected];

  //obtain the character indexed by this position
  next_misinfo_char =  misinfo_channels[misinfo_selected][index];

  //record next available position for similar
  index+=1;
  //applying bounds so as not to index outside the misinfo string
  if (index >= misinfo_len[misinfo_selected])
  {  index = 0;
  }//if

  //record this updated index
  misinfo_pos[misinfo_selected] = index;

  return next_misinfo_char;
}


//_____________________________________________________
//Sub-Program    : re_arrange_bits_of
//purpose        : re-arranges the bits of original_char
//               :
//return         : the re-arranged char
//               :
//parameters     : char original_char: the char to be re-arranged
```

```
//side-effects      : none
//
#ifdef BIT_SHUFFLE_MODE
//get the bit patterns
#include "bit_patt.h"
//in least significant (first char)
//to most significant bit(last char) order

byte bit_equivalents[]={1,2,4,8,16,32,64,128};

byte re_arrange_bits_of( byte original_char)
{
byte re_arranged_char = 0,
    current_bit;
static char current_pattern = 0;

 //for (each bit in the char) loop
 for( current_bit = 0; current_bit < 8; current_bit++)
 { //if( current_bit is set) then...
  if( original_char & bit_equivalents[current_bit])
  { //set its substituted bit in the re-arranged char
   re_arranged_char |= patterns[current_pattern];
  }//if

  //advance this shuffle pattern
  current_pattern+=1;
 }//for

 //placing bounds on the cycle of shuffle pattern
 //to keep the advance in cyclic fashion
 if( current_pattern >= TOTAL_PATTERNS )
 { current_pattern = 0;
 }//if

 return re_arranged_char;
}
#endif


//_____________________________________________________
//Sub-Program    : swap_consonant
//purpose        : if the_char passed in is a (swappable)
//               : consonant, randomly chooses a
//               : consonant with which it is swapped
//return         : if able to swap the_char, returns
//               : the substitute consonant; else
//               : returns the_char
//parameters     : the_char (in mode): a candidate character
//               : for swapping with another consonant
//side-effects   : none
//
#ifdef SWAP_CONSONANTS_MODE
char swap_consonant(char the_char)
{
char temp   = tolower( the_char),
    *index  = interchangeable_consonants;

 while(*index)
```

```
 { if( *index == temp)
   { temp = interchangeable_consonants[next_rand % consonants_length];
     return (is_upper(the_char) ? toupper(temp) : temp);
   }
   else
   {
     index++;
   }//if
 }//while

 return the_char;
}
#endif
```

8. Appendix B2: Source Code: Assembly code for PRNG.

```
;this code fragment demonstrates the bit-shift PRNG, optimised from Scheier's original C ;code for
use with the 68HC11 MPU.
;
parity_table
    fcb 0,128,128,0,128,0,0,128,128,0,0,128,0,128,128,0
    fcb 128,0,0,128,0,128,128,0,0,128,128,0,128,0,0,128
    fcb 128,0,0,128,0,128,128,0,0,128,128,0,128,0,0,128
    fcb 0,128,128,0,128,0,0,128,128,0,0,128,0,128,128,0
    fcb 128,0,0,128,0,128,128,0,0,128,128,0,128,0,0,128
    fcb 0,128,128,0,128,0,0,128,128,0,0,128,0,128,128,0
    fcb 0,128,128,0,128,0,0,128,128,0,0,128,0,128,128,0
    fcb 128,0,0,128,0,128,128,0,0,128,128,0,128,0,0,128

        ;=======================================

        ldab r_s1           ;get lowest byte
        andb #%01010111     ;isolate lowest byte's polynomial bits
        ldaa r_s4           ;get highest byte
                            ;  after this load the CCR (Flag register) has:
                            ;          - N bit is complement of Acc A MSB
                            ;          - Z bit set if Acc A are all reset
        bpl no_hibit        ;test the N bit in CCR
        orab #%00001000     ;set bit3 if highest byte msbit is set
                            ;  this diminishes the size of parity
                            ;  look up table to 128 bytes c.f. 256
                            ;  if we used the msbit
no_hibit
                            ;prepare to get the parity bit
                            ;from the parity look-up table
        ldx #parity_table   ;get table start
        abx                 ;add offset into parity-table

                            ;now the 4 byte shift for random seed update
                            ;
        lsra                ;first the highest byte
        oraa 0,x            ;or with parity of latest bit for insertion
        staa r_s4           ;and save this
        ror r_s3            ;rotate remainder in memory thru' carry flag
        ror r_s2            ;
        ror r_s1            ;
                            ;  before finally collecting next random
                            ;  bit to be
                            ;  returned from this iteration
        rol res1            ;first into the low byte
        rol res2            ;then on into the next higher byte
        rol res3            ;ditto
        rol res4            ;into most significant byte
                            ; note that res1 is aliased to next_rand in the C program

        ;================================
```

9. References

Baker, R. (1991). *Computer Security Handbook.* Blue Ridge Summit, PA: TAB Professional and Reference Books (McGraw Hill).

Bowcock, M. (1995). *Smart Cards in Australian Telecommunications Towards 2000.* Paper presented at the Australasian Smart Card Summit. 27-28 March 1995. Sydney Australia

Burton, S, Kaliski, Jr,. and Robshaw, M. (1996). Multiple Encryption: Weighing Security and Performance. *Dr Dobbs Journal.* Jan '96 pp. 123-127.

Caelli, W., Longley, D., and Dawson, E. (1994). Applications and Theory of Cryptography. In *Information Security Handbook.* Caelli, Longley and Shain. Basingstoke, UK: Macmillan Press., 1994, p 349

Cordonnier, V. (1995, June). Seminar on Smart-cards given at Edith Cowan University, Mount Lawley campus.

Dancer, H. (1995, May). Visa trials cash on a card in Australia. *Australian Personal Computer.*

De Schutter, B (1991) Trends in the Fight Against Computer-Related Delinquency. In Preneel, Govaerts & Vandewalle (Eds) *Computer Security and Industrial Cryptography: State of the Art and Evolution.* Berlin: Springer-Verlag.

Dellecave, T. (1995, June 26th) Red flag raised over standards. (Institute of Electric and Electronics Engineers 802.11 interoperability standard) *InformationWeek*

Dinnissen, P. (1995, March). Secure Chip Cost Breakthrough. *Smart Card News.*

Everyman's Encyclopaedia (1973) London: J.M. Dent & Sons Ltd.

Ferguson, N. (1994). Single Term Off-Line Coins. In Tor Helleseth

(Ed.), *Advances in Cryptology - Eurocrypt '93.* Germany: Springer-Verlag.

Ferreira, R. (1990). The Practical Application of State of the Art Security in Real Environments. In Seberry & Pieprzyk (Eds.), *Advances in Cryptology - AUSCRYPT '90.* Germany: Springer-Verlag.

Ford, W. (1994). *Computer Communications Security.* Englewood Cliffs, New Jersey: Prentice Hall.

Fox, B. (1993, August). Phantom Card Tricks. *New Scientist.* pp. 45-46.

Fumy, W. (1991). Local Area Network Security. In Preneel, Govaerts & Vandewalle (Eds) *Computer Security and Industrial Cryptography: State of the Art and Evolution.* Berlin: Springer-Verlag.

Greiner, G. & Reissberg, U. (1987). ECM Resistance in the HF Band by use of Adaptive Reaction and Frequency-hopping. *Military Technology (Electronics in Defence)* Vol 11 Issue 5.

Herskovitz, D. (1995, July). Wide, Wider, Widest. . *Journal of Electronic Defence.* Vol.18, No.7, pp. 51-58.

Herskovitz, D. (1995, June). Come, Whisper in My Ear. *Journal of Electronic Defence.* Vol.18, No.6, pp. 46 - 52.

Hilvert, J. (1997, June). What Price Your Private Life. *Information Age* pp. 22-25.

Hodgson, K. (1995, March). Smart Card's Advantage: Mobility, Security, Flexibility. *Security.* pp. 22-24.

Hoffman, T.(1995, March). Visa offers alternative to debit cards. *Computerworld* v29 n13

Hook, C. (1995, April). Peculiar problems associated with applying RF/ID. *Automatic I.D. News Europe.* pp. 25-27.

Hutcheon, A. (1992). Automated Teller Machines. In Jackson & Huskra (Eds.), *Computer Security Reference Book.* Boca Ranton, Florida: CRC Press, Inc.

Ingleby, M. (1988). Speech and image signals: recognition as classification under uncertainty. In Moscardini & Robinson (Eds) *Mathematical Modelling for Information Technology: Telecommunication reception, transmission and security.* Chichester: Ellis Horwood Limited.

ISO/IEC 7816-3 Part 3, (1994). ISO/IEC 7816-3. *Identification Cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols.* Geneva, Switzerland: ISO/IEC Copyright Office.

Janacek, G. & Lever, K. (1988). Ultra-Uniform Pseudorandom Number Algorithms for Cryptokey Generation. In Moscardini, A. & Robson, O. (Eds) *Mathematical Modelling for Information Technology, Telecommunication Reception, Transmission and Security.* Chichester: Ellis Horwood Ltd.

Johnson, J. & Tolly , K. (1995, July). Lab Test: Token Authentication Systems. *Australian Telecommunications.* pp. 76-80.

Kientzle, T. (1997, April). Algorithm Alley: Understanding CRCs. *Dr Dobbs Journal.* pp. 103-107

Kirkpatrick, S. (1995, April). Building Access: We take Visa, AmEx, MC... *Security*, pp. 30-32.

Landwehr, Heitmeyer & Mclean, (1984, August). A Security Model for Military Message Systems. *ACM Transactions on Computer Systems* 2(3) pp. 198-222.

Lathom-Sharpe, D. (1995). *Understanding What Smart Cards can do for You and Your Customers.* Paper presented at the Australasian Smart Card Summit. 27-28 March 1995. Sydney Australia.

Lindley, R.A. & Tweedle, J.K. (1995). *The Next Challenge: Designing Secure Multifunction Systems.* Paper presented at the Australasian Smart Card Summit. 27-28 March 1995. Sydney Australia

Longley, D. & Shain, M. (1989). *Data and Computer Security. Dictionary of Standards, Concepts and Terms.* Basingstoke: Macmillan Publishers Ltd.

Longley, D. (1994). Access Control. In Caelli, Longley & Shain, *Information Security Handbook.* Basingstoke, UK: Macmillan Press.

Longley, D., Dawson, E., & Caelli, W., (1994). Applications and Theory of Cryptography. In Caelli, Longley & Shain, *Information Security Handbook.* Basingstoke, UK: Macmillan Press.

Monod, E. (1990, April). France and Germany: Comparison of their Health Insurance Chip Cards and Implementation Strategies. *Smart Card Monthly* pp. 16-19.

Newton, H. (1994, Dec.). Feeding frenzy.(investing in telephone debit cards), *Teleconnect* v12 n12

Olsen, F. (1995, May). Encryption cards and key management fall into place. (the Department of Defense's Fortezza PC Card), *Government Computer News* v14 n9

Parker, D (1983). *Fighting Computer Crime.* New York: Schribner's Sons.

Paterson, M. (1991). Secure Single Chip Microcomputer Manufacture. In Chaum, D.(Ed.), *Smart Card 2000.* Holland: Elsevier Science Publishers B.V.

Phillips (1994). *DX Smart Card.* (Manufacturer's literature) Eindhoven, Holland: Phillips B.V.

Rigney, S. (1995). Spread the signal. *PC Magazine* v14 n1

Rivest, R., Shamir, A., & Adelman, L., (1978, Feb). A method for Obtaining Digital Signatures and Public key Cryptosystems. *Communications of the ACM.* Vol. 21(2), pp. 120-126.

Roberts, D. (1991). Evaluation Criteria for I.T. Security. In Preneel, Govaerts & Vandewalle (Eds) *Computer Security and Industrial Cryptography: State of the Art and Evolution*. Berlin: Springer-Verlag.

Scheier, B. (1992, Feb). Pseudo-Random Sequence Generator for 32-bit CPUs. *Dr Dobbs Journal*. p. 34

Seberry, J. and Pieprzk, J. (1989). *Cryptography: An Introduction to Computer Security*. Englewood Cliffs: Prentice Hall.

Sebesta, R.W. (1989). *Concepts of Programming Languages*. California: The Benjamin Cummings Publishing Company Inc.

Seidman, S. (1995, April). News. *Smart Card Monthly*. pp. 10-15.

Shannon, C. (1949). Communication Theory of Secrecy Systems, *Bell Systems Technical Journal*. Vol. 29, pp 656-715.

Shiflet , A. (1987). *Discrete Mathematics for Computer Science*. New York, West Publishing Company.

Smulders, P. (1990, February). The Threat of Information Theft by Reception of Electromagnetic Radiation from RS-232 Cables. *Computers and Security* pp. 53-58.

Telefunken System Technik. (1993). *Electronic Warfare of the Land Forces*. Product information from Telefunken System Technik GMBH, Empfanger und Peiler, Sedanstrasse 10, D-7900 Ulm, Germany.

TIRIS (1994). *Texas Instruments Registration and Identification Systems. TIRIS - Overview of Technology, Products and Applications*. Austin TX: Texas Instruments.

Torrieri, D. (1992). *Principles of Secure Communication Systems*. London: Artech House.

Townend, R.(1995). *Chip Card Vision - The Viewpoint of Mastercard International*. Paper presented at the Australasian Smart Card Summit. 27-28 March 1995. Sydney Australia.

Twentyman, J. (1997, June). Safe and Secure. *Information Age* pp. 32-36.

Van Order (1995, July). Military Logistics Goes High Tech. *SOLEtter, A publication of the Society of Logistics Engineers.* Vol. 30, Issue 5. Hyattsville, MD 20785, USA.

Vandewalle, J., Govaerts, R., & Preneel, B. (1991). Technical Approaches to Thwart Computer Fraud. In Preneel, Govaerts & Vandewalle (Eds) *Computer Security and Industrial Cryptography: State of the Art and Evolution*. Berlin: Springer-Verlag.

Vedder, K. (1992). Smart Cards. *Proceedings CompEuro92*. Los Alamitos: IEEE Computer Society Press. pp. 630-635.

Vereen, L. (1996). Software Tools for Embedded Systems Development, *Embedded Systems Programming*, Vol. 9. No. 10., pp19-57.

Verschuren, Govaerts, R. and Vandewalle, J. (1991). ISO-OSI Security Architecture. In Preneel, Govaerts & Vandewalle (Eds) *Computer Security and Industrial Cryptography: State of the Art and Evolution*. Berlin: Springer-Verlag.