Reliable and Secure Information Storage on a Capacitive Memory Card

by

Alon Gendel

B.Sc., University of Tel-Aviv, 1996

A thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE

In the Department of Electrical and Computer Engineering

We accept this thesis as conforming to the required standard

[redacted]

Dr. P. Agathoklis, Supervisor (Department of Electrical and Computer Engineering)

[redacted]

Dr F. El Guibaly, Departmental Member (Department of Electrical and Computer Engineering)

[redacted]

Dr. H. A. Müller, Outside Member (Department of Computer Science)

[redacted]

Dr. Z. Dong, External Examiner (Department of Mechanical Engineering)

Supervisor: Dr. P. Agathoklis

ABSTRACT

The Coincard is a capacitive, stored value, plastic memory card, which is used as a low-cost, portable, disposable information storage medium. Coincards are read using specialized readers using zero-proximity, capacitive coupling. The card's typical applications are in the transit, vending, and identity card markets. A card used in such applications, being nothing more than a thin, flat piece of plastic, is subject to damage caused by scratches or bends. Depending on the application, it might carry a monetary value, which makes it a target for counterfeit attempts. This work presents a system solution for the secure and reliable storage of information on the card. The security aspect of the solution incorporates private-key encryption and authenticity validation methods. Card data is encrypted, and signed using an authentication code, making fraud attempts either infeasible or impractical. The error correction scheme uses a unique error correcting code and a card information placement scheme that make it more resistant to physical damage. Both the security and reliability objectives are achieved without adding to the cost of the card, and without inflicting inhibitive costs on the card's reader software or hardware.

Examiners:

[redacted]

Dr. P. Agathoklis, Supervisor (Department of Electrical and Computer Engineering)

[redacted]

Dr F. El Guibaly, Departmental Member (Department of Electrical and Computer Engineering)

[redacted]

Dr. H. A. Müller, Outside Member (Department of Computer Science)

[redacted]

Dr. Z. Dong, External Examiner (Department of Mechanical Engineering)

ABBREVIATIONS

ASIC - Application Specific Integrated Circuit

AUED - All Unidirectional Error Detecting

BCD - Binary Coded Decimal

BCH - Bose, Chaudhuri, Hocquenghem [codes]

BSC - Binary Symmetric Channel

CRC - Cyclic Redundancy Check

DAC - Data Authentication Code

EC - Error Correcting

ECC - Error Correcting Code

GF - Galois Field

IC - Integrated Circuit

ISO - International Standards Organization

I/O - Input/Output

MAC - Message Authentication Code

PCB - Printed Circuit Board

PROM - Programmable Read Only Memory

TEA - Tiny Encryption Algorithm

WORM - Write Once, Read Many

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Card systems and the Coincard system

1.1.1 Motivation

Identification and vending cards have been in use for decades and are a common sight. Their typical role is to identify their carrier as entitled to perform some kind of action or transaction. Driver licenses entitle their holder to drive a vehicle and bus tickets entitle a person to ride a bus. At the other end of the spectrum are electronic-purse smart cards, which encapsulate a complete microprocessor with its own memory and operating system, function as a safe alternative to carrying cash and performing a variety of transactions. Machine-readable cards can store information using mechanical, optical, magnetic, or capacitive means, or inside an embedded silicon chip.

Memory cards are information storage cards that usually contain stored value that the user can spend in a retail transaction, for public transport, or for a community service. These stored-value cards contain some form of nonvolatile memory and are typically discarded when the balance runs out. Using memory cards has distinct benefits over carrying cash. From the users' point of view, cards are extremely thin and light and can be carried in any pocket or purse. There is no need to carry small change or cash for performing a transaction. From the vendor or service provider point of view, the need to store, handle and move cash is eliminated, and there are no coin boxes that serve as a target for thieves, thus the cost per transaction can be lower than that of a cash transaction. The possibility of identifying the card owner enables a variety of enhancements to be incorporated into a card system, such as collecting transaction data for statistical analysis, offering special

promotions for card holders, using the card as an employee card or medical record access card, etc.

1.1.2 Typical application – Mass Transit Fare System

A typical application of a memory card is a metropolitan mass-transit fare system. Many types of cards can be issued, including pre-paid ride cards containing 5, 10, or 20 rides, daily or monthly bus passes, concession cards, etc. Cards can be used in several related systems, performing different tasks. As an example, a single card may allow the holder to board any bus, train or ferry in a particular day, and be entitled to a prepaid amount of refreshments served on board.

As opposed to chip-cards or smart cards, which contain an encapsulated microprocessor, memory cards can be made extremely cheap and disposable. For the aforementioned type of applications, a small memory capacity is usually of no concern, since a disposable information storage medium of this sort is not likely to require a large amount of data storage in the first place. The most important card qualities sought after in this kind of application are its manufacturing costs, its susceptibility to fraudulent use or re-loading, and its reliability and durability.

1.1.3 The Coincard system

The Coincard (see Figure 1 Figure 2) is an example for such a small-memory, disposable transaction card. The system is comprised of a central manufacturing plant, capable of producing millions of cards; a distribution mechanism, through which cards are distributed or sold to customers; and a plurality of card readers associated with the application. At the heart of the system is the card itself, a memory card with a relatively small capacity, which is inexpensive to produce, and non-reloadable. A database that keeps track of cards,

transactions or any other application information can also be incorporated as part of the system.

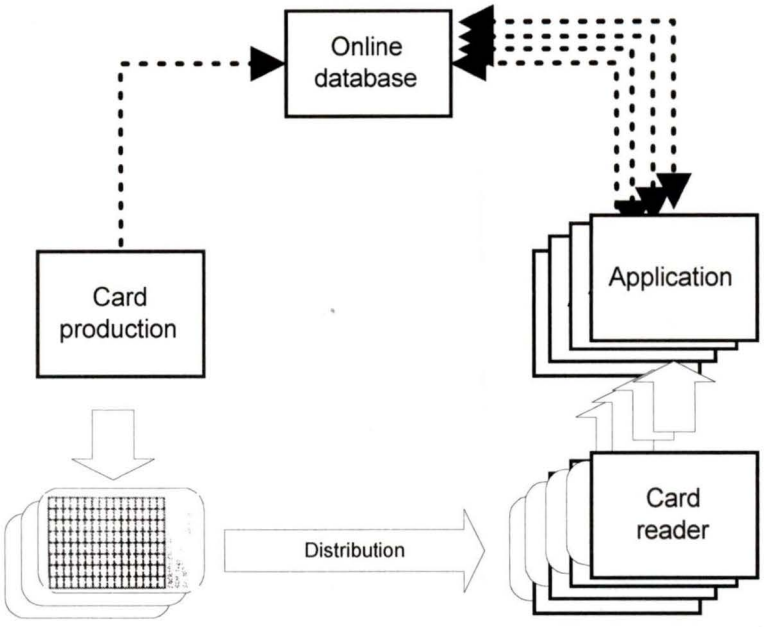


Figure 1: The Coincard system - schematic diagram

All cards are read using identical card readers, typically motorized insertion readers which may include a driver's display, a small printer, etc., and may also be connected to a central database, either online or through periodic updates.

The cards and readers themselves are intended to be generic, in such a way that multiple vendors with different applications can all use the same card technology, and the same card readers embedded in their applications. By including an online database, or a periodically updateable database, it is possible to enhance the system capabilities according to the target application. The database can keep track of card usage, record each transaction's time and place, assist in the identification or removal of stolen or fraudulent cards from the system, and support the generation of comprehensive usage statistics reports.

A similar, competing technology of the Coincard is magnetic-stripe cards [3]. Stripes of a magnetic resin can be bonded to a paper, plasticized paper or plastic media. Data is encoded in the magnetic stripe using a write head, a device that induces a magnetic field while being moved in close proximity (or contact) with the magnetic stripe surface. By altering the magnetic polarization of resin particles, values or "0" or "1" can be encoded. Data can be stored in one or more tracks along the magnetic stripe. Reading the card involves moving the card's magnetic stripe against a read head, thereby inducing electric currents that can be picked up, amplified and measured. A typical capacity of a magnetic stripe card (ISO standard 7811 [3]) is between 15 and 42 BCD digits per inch. The movement speed need not be constant, and moving the card can be done manually (swipe-cards) or mechanically. Magnetic stripe paper tickets offer a low-cost disposable storage medium, matched with an inexpensive reader. Although not very durable, and prone to fraudulent reloading, magnetic stripe cards have been used for many years in the mass transit and vending industries.

1.2 Coincard technology overview

The card itself (depicted in Figure 2) is a credit-card size plastic card, measuring 250μm in thickness. It is composed of two layers of a non-conductive polymer, and a thin layer of metal encapsulated in-between. The card data is etched on it using a laser beam, and is read by inserting the card into the specially designed reader.

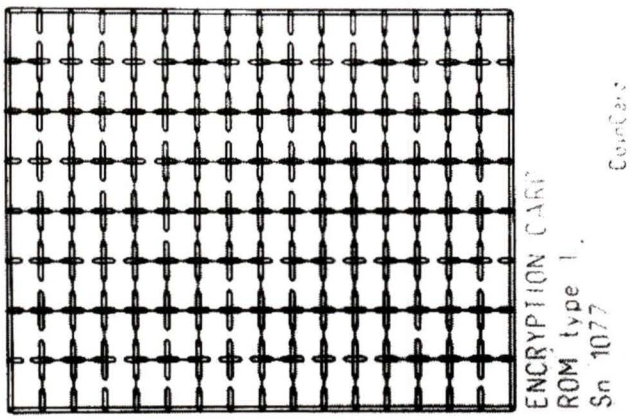


Figure 2: Coincard photo, showing the pads as etched in the metal layer by the laser beam

Binary data is stored on the card as the presence or absence of a low-impedance conductivity path between two adjacent "pads". As seen in Figure 2, an array of 8 by 16 pads is etched on the card, and the borders between adjacent pads differ- some are clearly adjoined together, while others are separated by a thin line. The reading of the card data is accomplished by accurately placing the card on a PCB, where a matching pattern of pads comes in contact with the card surface. As a specific PCB pad is energized with a high-frequency signal, the adjoining PCB pad senses the level of signal received from the adjoining pad on the card. If the pads are separated by the laser-beam etched line, there is no metal conductivity path between the two pads, and the measured level of signal will be small, mainly due to crosstalk and leakage. If, however, the etching laser beam left the metal layer intact across the pad "border", a large proportion of the signal will be picked up. Thus a border between two adjacent pads can be simply referred to as a "bit". The Coincard is patented technology, and is comprehensively described in [1] and [2].

Additionally, a special variant of pad border can be etched in the card, in such a way that it does not fully disconnect the two pads metal surfaces, and a high-impedance conductivity

path still exists. This type of etching, or "fuse", can be read normally, as any other bit, but it can also be burned – by driving the two pads with a higher-frequency signal, it is possible to heat up the fuse between them until it melts, and the metal permanently disconnects. This type of fuse is known as a WORM (Write Once Read Many) fuse. It enables a wide range of stored-value applications, where a specific cash (or token) amount can be deducted from the card with each and every transaction.

The entire process of fully inserting the card into the reader, reading and analyzing its contents, and possibly deducting one or more tokens takes less than one second; this is approximately the upper limit for a transaction in a typical application; a person boarding a bus will find a wait of more than one second noticeable.

The current pad allocation is summarized in the following list:

- A "card ID", or card type field. It is comprised of 20 fuses, divided into 5 pairs and 5 "backup pairs", each pair can either take the value "01" *or* "10". Thus the "card ID" is a 5-bit field, where each bit is encoded using 4 fuses, and is capable of correcting 1 fuse error per bit.
- A 21-bit card serial number, capable of representing $2^{21} = 2{,}097{,}152$ distinct cards.
- An 18-bit "Vendor ID" field, capable of representing $2^{18} = 262{,}144$ different "Vendors".
- An array of 36 WORM fuses.

1.3 Motivation

The current state of the Coincard technology is suitable for several applications, and was used successfully in such fields as mass transit and vending. In order to expand the range

of possible applications for the card, enhancements to the product's capabilities and performance are sought after. This work focuses on three main issues, summarized below:

Card data capacity utilization: The current data layout does not take full advantage of the card's capacity. Only 59 out of the available 232 fuses (bits) are being used for data storage.

Error correction capability: Of the 59 fuses used, 39 have no error correction capability, and 20 fuses are used to encode 5 "bits" of information with a limited error correction capability.

Provisions for securing card information: the existing layout of data on the card provides no security at all. Binary data is placed on the card as is, without any encryption or data hiding being employed.

1.4 Thesis work outline

In Chapter 2, an overview of cryptography and security is presented, and important concepts that are part of any secure communication system are formally defined. Since the Coincard is essentially a "channel" through which information is transferred from a "sender" application to a "receiver" application, general concepts from the realm of cryptography can be applied to the Coincard platform as well.

The issue of error correcting codes has been studied extensively and an overview of error correcting codes is outlined in Chapter 3. The basic concepts of information theory and coding are first presented, followed by the specific properties and characteristics of the Coincard "channel". Some error correcting code construction methods for this type of channel are presented.

Chapter 4 covers the solution development process, from the formal definitions of the application requirements and constraints, through the selection of the best cryptographic methods to be used and error correction scheme to be employed. An analysis of the Coincard platform shows that its constraints inhibit the use of any methods already in existence, since they impose some unavoidable irregularities on any kind of solution (for example, no known error correcting code construction method can be used).

An analysis of the solution performance metrics is given in Chapter 5. The robustness of the security mechanisms, and the performance of the error correction scheme are studied and presented. Simulation of the ECC performance, performed by inflicting a range of random errors and clustered (physically adjacent) errors on simulated "cards", gives good approximations to the overall card reliability expected in any future application. A more theoretical analysis of the security methods employed provides an insight into the likelihood of successful fraud committed against a Coincard system. In section 5.4 an outline is drawn for a practical use of the Coincard in a mass transit fare collection system.

Finally, several points that are closely related to this work, are discussed in Chapter 6 as items for future work. These topics are not part of the Coincard platform itself and were not implemented, but enhance the capabilities and suitability of the Coincard application for a wider range of applications. The topics include solution suggestions for such issues as secret key management, and secure card production. Another future work item relates to an alternative card geometry, which subdivides the card surface to hexagons instead of rectangles. Hexagonal and other possible geometries were outlined in the original Coincard patent, but were never implemented. Any such implementation would benefit from the error correction and security contributions of this work.

2 CRYPTOGRAPHY AND SECURITY

2.1 Introduction

This chapter gives an overview of cryptographic terms, concepts and applications. This relatively new science is part of any modern computer or communication security. As pointed out, the Coincard can be viewed as the "medium", or channel, through which data is transmitted from the sender (the card etching machine) to the receiver (the card reader application). This equivalence between the Coincard and an arbitrary digital communication channel means that cryptographic methods are applicable in the Coincard case. This chapter covers those methods and security issues that are relevant to the Coincard system, and gives formal definitions to concepts such as a cipher, the several types of attacks against a cipher, and the possible consequences of a successful attack. Other methods for assuring data integrity and authentication are also presented.

A discussion of the wider concept of security follows the cryptography overview, and it focuses on the security of microcontroller-based applications and card systems, as they are the most relevant for this work.

2.2 Cryptography

Cryptography is defined as the practice and study of encryption and decryption - encoding data so that only specific individuals can decode it. The goals of cryptography extend further than simply obscuring the contents of a message, and can be summarized in these four objectives:

1. Secrecy (also privacy, confidentiality): the contents of the information are kept secret from anyone but those who are authorized to receive it.
2. Integrity: the information being sent cannot be changed or altered in any way without the receiving party detecting it.
3. Authentication: the parties performing the communication can identify each other, making impersonation impossible. Furthermore, the information being transferred can be authenticated as to its origin and content.
4. Non-repudiation: a party engaged in the communication process cannot deny previous communication activities or commitments.

While cryptography has a long history before the age of digital communication, the overview presented here will concentrate on its applications in computing and digital communication.

2.3 Terminology and concepts

A message is referred to as a *plaintext.* The process of hiding and obscuring a message's contents is referred to as *encryption.* An encrypted plaintext is called a *ciphertext*, and the procedure used to convert a ciphertext back to plaintext is called *decryption.* While cryptography is the science of securing messages, and deals with designing and analyzing cryptographic systems, *Cryptanalysis* is the art and science of breaking ciphertexts and revealing a message's contents.

A *cryptographic algorithm* (also referred to as a *cipher)* is the mathematical function that is used for encryption and decryption. Cryptographic algorithms can rely on the specifics of the algorithm for security, or they may rely on a secret *key* (denoted by $\boldsymbol{k}$). The key is any

one of a large number of possible values, which form a *keyspace K.* Both the encryption and the decryption algorithms use a specific key (or a related pair of keys) to encrypt and decrypt messages. If the security of the algorithm relies only on keeping the inner workings of the algorithm secret, the algorithm is called a *restricted* algorithm. As an example, a simple restricted text encryption algorithm might consist of a secretive sequence of alphabetic substitutions and letter position shuffling. Since restricted algorithms have some major drawbacks, keys are normally relied upon for the security of the algorithms.

A *cryptographic system* is the cryptographic algorithm plus the set of all possible plaintexts, ciphertexts, and keys.

In the following definitions, the encryption and decryption are described as functions operating on texts (messages), where the texts are strings of symbols that form finite spaces.

Plaintext is denoted by M. M consists of strings of symbols from an alphabet A, which in computer communications will usually be the alphabet $A=\{0,1\}$. A ciphertext is denoted by C, which consists of strings of symbols from an alphabet (not necessarily the same alphabet used to represent M).

The process of encryption can be denoted by the transformation

$$E(M) = C \tag{2.1}$$

i.e., the encryption function E operates on M to obtain C. The reverse process of decryption, D, operates on C to obtain M:

$$D(C) = M \tag{2.2}$$

It follows that:

$$D(E(M)) = M \tag{2.3}$$

if a key ***k*** is being used, the notation becomes:

$$E_k(M) = C \tag{2.4}$$

$$D_k(C) = M \tag{2.5}$$

and

$$D_k(E_k(M)) = M \tag{2.6}$$

Data content integrity is defined as the state of data that has not been altered in any manner since the time it was created, or transmitted. Data content integrity can be affected by the insertion or deletion of data bits, reordering of data bits, inverting data bits, or any combination of those operations. A broader definition is that of *data origin authentication*, which refers to an authentication method where a specific party can be identified to be the original source of a particular piece of data, at some point of time in the past.

2.4 Cryptanalysis

Many different types of attacks against cryptographic systems have been devised. These can be divided into two sub-categories:

1. *Passive attacks,* take place in a scenario where an adversary only eavesdrops to the communication channel traffic. The secrecy of the data is thus threatened.

2. *Active attacks* refer to a scenario where an adversary has the ability to alter or tamper with the transmission on the communication channel, threatening not only the secrecy of the information but also the data integrity and authentication.

Attacks against encryption schemes can be subcategorized as below:

1. *Ciphertext-only attack*: The cryptanalyst tries to recover the decryption key or the plaintext of messages by only observing the ciphertext of several messages.
2. *Known-plaintext* attack: The cryptanalyst has access to the plaintext as well as to the corresponding ciphertext of a number of messages.
3. *Chosen-plaintext attack*: The cryptanalyst further has the ability to choose the plaintext that is being encrypted and observe the corresponding ciphertext. This attack is more powerful than (2) since the adversary has the ability to choose specific plaintexts to encrypt, ones that might reveal more information about the key.
4. *Adaptive chosen-plaintext* attack: this is a chosen-plaintext attack where the adversary can modify his choice of plaintext to encrypt based on the results of previous encryption.
5. *Chosen-ciphertext* attack: The adversary can select different ciphertexts to be decrypted and observe the corresponding plaintexts, in an attempt to recover the key.
6. *Adaptive chosen-ciphertext attack*: a chosen-ciphertext attack where the adversary can select the ciphertext to decrypt according to the resulting plaintext from previous decryptions.

An additional type of attack is a *Chosen-key attack,* where the cryptanalyst has some information about the relationships between different keys.

Successful attacks can be categorized according to the level of their severity:

1. *Total break*: the adversary recovers the key k, such that $D_k(C) = M$.

2. *Global deduction*: the adversary finds an alternate algorithm, which is equivalent to $D_k(C)$, without knowing k.

3. *Local deduction*: the adversary recovers the plaintext of a particular ciphertext.

4. *Information deduction*: the adversary gains some knowledge about the cryptographic system, for example, a part of a key, or information about the structure of the plaintext, etc.

2.5 Symmetric-Key encryption

Symmetric-key encryption, also known as single-key or secret-key encryption, is a scheme where for each associated encryption/decryption key pair, it is "easy" to determine one key from the other. In most practical systems, the keys are identical.

In symmetric-key cryptography, the two sides engaged in the message transfer must agree beforehand on the key to be used. The key therefore must be exchanged in secret; otherwise the messages will not be secure (see Figure 3). The problem of finding a secure way to exchange the secret key is known as the *key distribution problem* [15].

There are two distinguished classes of symmetric-key encryption schemes: *stream ciphers* and *block ciphers*.

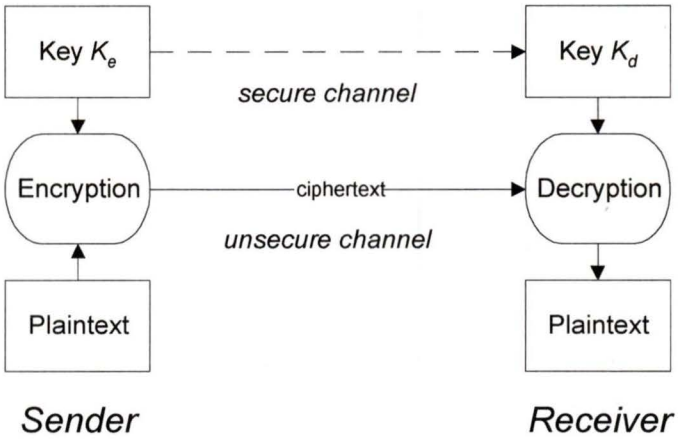


Figure 3: Symmetric-key cryptography system schematic diagram.

2.5.1 Block ciphers

Block ciphers are cryptographic algorithms that operate on the plaintext by breaking it up into segments (or blocks) of equal length, and encrypting it one block at a time. The length of the resulting ciphertext block is the same as the length of the plaintext block, and is referred to as *block size*.

2.5.2 Stream ciphers

Stream ciphers can be viewed as block ciphers that operate on a block size of 1 (one symbol at a time).

2.6 Public-key cryptography

Public key cryptography uses two keys: one is public, and the other is kept secret, and it is computationally hard to deduce the decryption key K_d from the encryption key K_e. Any

party can encrypt a message using the public key, but only a party holding the private key can decrypt the message.

Public key cryptography solves the key distribution problem, inherent in symmetric-key systems. There is no need for a secure channel where keys are shared, since the sending party simply publishes her public key, and anyone can then encrypt messages and send them using an insecure channel. A shortcoming of public key cryptography is its performance – public key cryptographic algorithms are typically 1000 times slower than secret-key algorithms [4].

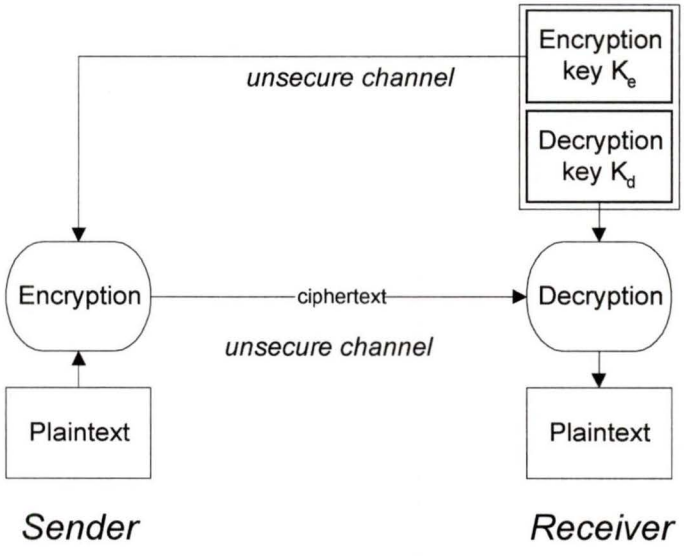


Figure 4: Public-key cryptography system schematic diagram.

2.7 One-way functions and hash functions

One-way functions [11] are functions that are easy to compute, but significantly harder to reverse (i.e., $f(x)$ is easy to compute from x, but it is hard to obtain x given $f(x)$). Mathematically, there is no proof that one-way functions can be constructed or even that

they actually exist. However, many functions appear to be one-way (as an analogy, calculating x^2 is "easy", while it is much harder to calculate its inverse, $\sqrt{x}$), and can thus be referred to as "candidate" one-way functions. Further analysis of the computational complexity of calculating one-way functions is given in [5].

A *trapdoor* one-way function [11] is a one-way function where the inverse direction is easy, given a certain piece of information, known as the trapdoor, but difficult otherwise. A public–key encryption scheme also operates on the trap-door function principle: the public key is equivalent to specifying the function $f(x)$. Without the trapdoor (secret key) information, it is computationally infeasible to decrypt the original plaintext from the ciphertext.

A *hash function* H is a transformation that takes an arbitrary-length message m as input and returns a fixed-length string, called the hash value $h = H(m)$. Hash functions are useful in cryptography, since the hash value h serves as a representative image (also referred to as a *digital fingerprint*, or *message digest*) of a particular string, and thus can be practically uniquely identifiable with that string – assuming it meets certain requirements as detailed below.

A *one-way hash function* (first defined by Merkle [10]) is a hash function that fulfills the following requirement:

1. Given a hash value h, it is computationally infeasible to find any input x such that $H(x) = h$.

A *weakly collision-free hash function* further fulfills the second requirement:

2. Given an input x, it is computationally infeasible to find y such that $H(x) = H(y)$.

And a *strongly collision-free hash function* further fulfills the third requirement:

3. It is computationally infeasible to find two inputs x and y such that $H(x) = H(y)$ (here there is no pre-selection of any input).

The security of a hash function is due to its one-way property, thus the hash algorithm itself need not be kept secret.

Since hash functions are generally faster than digital signature algorithms, it is typical to compute the digital signature to some document by computing the signature on the document's hash value, which is small compared to the document itself. This saves the more complex and costly signing of a large document, which might involve breaking up the document into numerous blocks of data and signing each one separately. Additionally, a digest can be made public without revealing the contents of the document from which it is derived.

Hash functions can also be used for verifying data integrity. After computing the hash value for some data, the hash value is stored and kept as reference. At a later time, when the data integrity needs to be checked, the hash value of the data is computed in the same manner, and compared against the stored reference hash value. If they are identical, this implies that the data has not been altered.

2.7.1 Hash functions using symmetric block ciphers

The rationale behind using a block cipher as a hash function, is that if the block cipher is secure, so will be the hash function. Additionally, if in a certain application a block cipher

infrastructure already exists (either in hardware or in software), then the additional hash functionality can be achieved using this already available infrastructure. Hash functions that use block ciphers can be divided into two subgroups – cases where the hash value length is equal to the block cipher's block size, and cases where the hash value is twice the block length.

An important parameter of a hash function that uses a block cipher is its *rate,* which is defined as the number of blocks processed per encryption. The higher the rate – the faster the hash function is.

A basic hash algorithm that uses a block cipher, known as Davies-Meyer [12], is depicted in Figure 5.

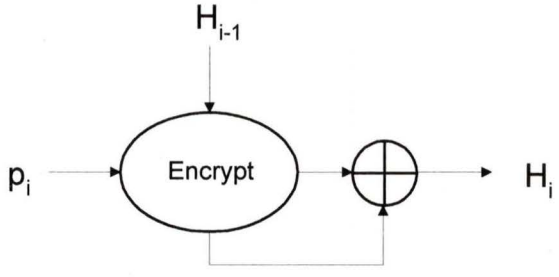


Figure 5: Davies-Meyer hash algorithm, a single length hash function using a block cipher.

The input is a plaintext bit-string p_i of length n, and the output is the n-bit hash value of p_i. The algorithm is comprised of the following steps:

1. A constant initial value I_0 is selected

2. The input p_i is divided into strings of length k, where k is the length of the cipher key, and the last string is padded if necessary. The strings are denoted as $p_0, p_1, \ldots, p_t$.

3. The output is denoted as H_t, and is calculated by:

$$H_0 = I_0 \quad H_i = E_{key=p_i}(H_{i-1}) \oplus H_{i-1}, \quad 1 \le i \le t \tag{2.7}$$

2.8 Message authentication codes (MACs)

A message authentication code (MAC) (or a data authentication code, DAC) is a one-way, key-dependent hash function. A unique value, or tag, is sent alongside the message and serves as a verifier to its authenticity; only a party that knows the secret key can re-generate the hash value from the data and thus authenticate the message, since the hash value h is a function of both the message and the key. MACs are therefore useful in providing authenticity (data integrity) without necessarily providing privacy. There are different ways to construct MAC schemes, and they can be categorized according to the method used to create them: unconditionally secure[1], hash function-based, stream cipher-based, or block cipher-based.

A trivial way to construct a MAC from a hash function is to encrypt the hash value h using some symmetric-key encryption algorithm.

---

[1] Unconditional security of a cryptographic method (or, for an encryption algorithm, *perfect secrecy*) means that the difficulty of defeating it can be shown to be equivalent to solving a well-known and supposedly "hard" problem (typically a number-theoretic problem). The concept was first introduced by Shannon [17].

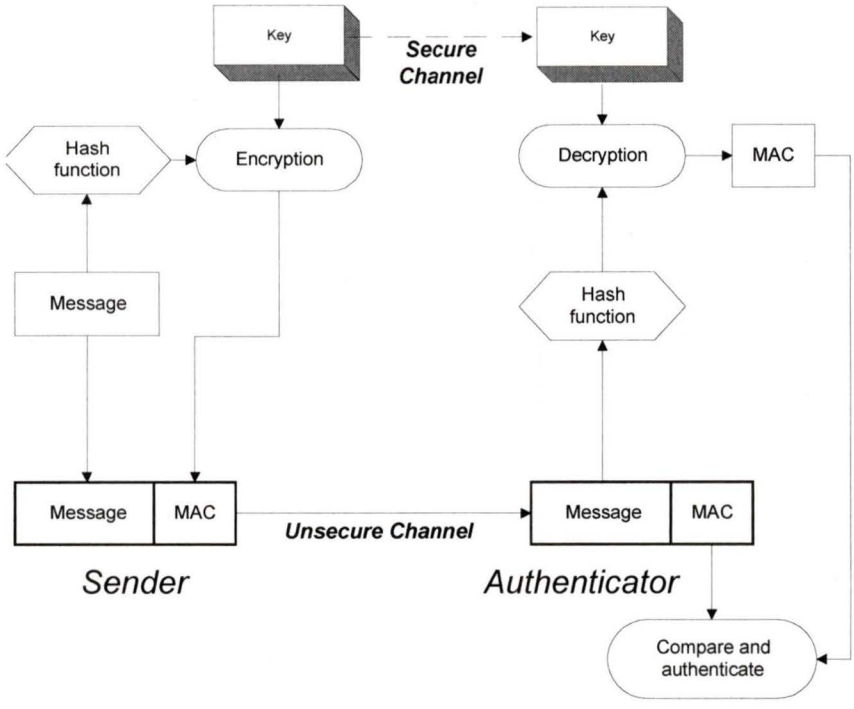


Figure 6: Sample data integrity verification scheme using hash functions.

It is worthwhile noting here that encryption alone cannot guarantee the authenticity or the integrity of data. An adversary might still be able to effectively manipulate encrypted messages in some way (for example, re-ordering of message blocks). When considering the encryption of data that does not have any redundancy in it (for example, a pseudo-randomly generated key), it is not possible to distinguish the correctly decrypted plaintext from a decryption of a ciphertext that was altered in some way - in both cases the message will look pseudo-random, or "meaningless".

The properties of a MAC can be summarized as follows:

1. The MAC generation is straightforward: for a hash function H, a key value k and any input x, the result $H(x,k) = h$ is easy to derive.

2. $H(x,k)$ maps x, a string of arbitrary length, to h, which is a string of fixed length.
3. It is computationally infeasible to compute $H(x,k)$ given any number of plaintext-MAC pairs $\{x_i, H(x_i,k)\}$.

Similar to attacks against encryption schemes, attacks on MACs can be categorized into the following types:

1. Known-plaintext attacks: where one or more pairs of plaintext and MAC, $\{x_i, H(x_i,k)\}$, are available to the adversary.
2. Chosen-plaintext attacks: where the adversary has the ability to choose the plaintext x_i and then obtain its MAC $H(x_i,k)$.
3. Adaptive chosen-plaintext attacks: where the adversary can modify his choice of plaintext to calculate the MAC based on the results of previous MAC results.

A distinction should be made between the difficulties of recovering the key k, and of forging a MAC without any knowledge of the key – the non-recoverability of k does not imply the MAC of a new plaintext x_j is not forgeable.

MAC forgery for an arbitrary input plaintext x_j, without any knowledge of the key k, requires some mechanism for validating MACs – either using a set of valid plaintexts and their matching MACS, or by using a "black box" which produces MACs for arbitrary plaintext inputs.

Recovery of the key used to produce the MAC means total break. It is also possible that an adversary can derive MACs of some plaintexts, but without the ability to arbitrarily choose those plaintexts.

2.9 Digital signatures

The concept of digital signatures was first introduced by Diffie and Hellman [11], and Merkle [10]. A digital signature is a number (or string), which is dependent of some secret known only to the signer, as well as on the contents of the document or message being signed. The act of signing involves binding a signer's *identity* to some plaintext. Unlike a handwritten signature, a digital signature is plaintext-dependent. Following the same trapdoor principle, the signing process is equivalent to the inverse calculation of x given $f(x)$, and the signature verification process is equivalent to the forward calculation of $f(x)$ given x.

The goals of a digital signature can be summarized in the following list:

1. Authenticity: the signature indicates that the signer indeed signed the document.
2. The signature cannot be forged: the signature is a proof that the signer (and no one else) signed the document.
3. The signature is not re-usable: the signature is part of the document, and cannot be transferred and used on a different document.
4. Integrity: after the document has been signed, it cannot be altered.
5. The signature cannot be repudiated: once a document has been signed, the signer cannot later claim that he or she did not sign it; it should be easy to verify the authenticity of the signature without revealing the signer's secret.

2.9.1 Digital signature formal definition

By using the following notation:

M - a set of plaintexts (or messages) that can be signed

S - a set of elements called the signature space

S_A - a *signing* transformation, which is defined as a transformation from the set M to the set S. The signer A keeps this transformation secret.

V_A - a *verification* transformation, is a transformation from the set $M \times S$ (all the pairs (m,s) where $s \in S, m \in M$) to the set $\{false, true\}$. The transformation V_A is publicly known and used to verify A's signatures.

A *digital signature scheme* can be described as the pair of transformations S_A and V_A.

The signing procedure simply involves computing the transformation $s = S_A(m)$ and sending the pair (m,s) to the receiver. The verification procedure consists of computing $u = V_A(m,s)$. If $u = true$ then the signature is accepted; otherwise, it is rejected.

It must hold true that it is computationally infeasible for any entity other than the signer to find any pair (m,s) such that $V_A(m,s) = true$.

Digital signature schemes can be classified into two groups:

1. *Digital signature schemes with appendix* (first described in [10]), which require the original plaintext as the input to the verification algorithm.

2. *Digital signature schemes with message recovery* [11], where the original plaintext is not required as the input to the signature verification algorithm; in those schemes, the original plaintext is recovered from the signature itself.

There are public-key algorithms that can be used as digital signature algorithms. A basic protocol would include the following steps:

1. The sender encrypts the message with his private key, thus signing it, and sends it to the receiver.
2. The receiver decrypts the message using the sender's public key, thus verifying the signature.

This simple scheme satisfies all the five goals of a digital signature:

1. The signature is authentic: whoever decrypts the message knows that the sender signed it
2. The signature is unforgeable: only the sender (signer) knows his private key.
3. The signature is non-reusable: it is created as a function of the message, and cannot be used on another message.
4. The signature is unalterable: if the message is altered in any way, it cannot be verified using the signature anymore.
5. The signature cannot be repudiated: unless the private key is revealed, the action of signing the document could not have been done by anyone else but the signer.

A comprehensive survey of digital signature algorithms can be found in [4].

2.10 Block-cipher design primitives

Most modern block cipher systems apply a number of iterations, or "rounds", in succession, to encrypt plaintext. Each round is constructed from a set of similar or identical operations, which add both *confusion* and *diffusion* to the encryption.

2.10.1 Diffusion

Diffusion spreads the influence of plaintext symbols or key symbols over as much of the ciphertext as possible. *Overall Diffusion* is an ideal block cipher where a change of a single plaintext bit will change every ciphertext bit with probability 0.5. In practice, a good block cipher will approach this ideal. This means that about half of the output bits should change for any possible change to the input block.

2.10.2 Confusion

A mechanism that changes the correspondence between input and output values, and thus facilitates the hiding of any relationship between plaintext, ciphertext and key material. For example, performing an XOR of the plaintext with a pseudo-random data sequence (key) will result in a pseudo-random looking ciphertext.

2.10.3 Substitution and transposition

A simple *substitution* involves mapping a plaintext alphabet into a ciphertext alphabet. A more complex substitution scheme would use different alphabet mappings on different portions of the plaintext. A *transposition* is simply the exchange in position of two elements. A substitution adds confusion in the process of encryption, while a transposition adds diffusion.

2.10.4 Product ciphers

A product cipher is a cipher that combines two or more transformations, in such a way that the resulting cipher is more secure than the individual transformations.

2.10.5 Feistel networks

In a Feistel network [14], each block of data is divided into two halves: L and R. First, the current R is assigned to the next L. Then, the R part is transformed and the result is XOR'd with L and the result is assigned to the next R. This constitutes a single iteration; the process is typically iterated many times. The Feistel construction can be represented in the following form:

$$L_i = R_{i-1} \tag{2.8}$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i) \tag{2.9}$$

A cipher that uses a Feistel network is guaranteed to be invertible: it does not matter what the function f is; to reverse the process, the same transformation is simply applied again. This will undo the effect of the original exclusive-OR. One disadvantage of the Feistel network mechanism is that the diffusion depends on f, and there is no guarantee of overall diffusion.

2.11 TEA Tiny Encryption Algorithm

As an example for a practical encryption algorithm, the TEA algorithm is presented. TEA has been developed by Wheeler and Needham of Cambridge University, England,

```
void code(long* v, long* k) {
unsigned long y=v[0],z=v[1], sum=0, /* set up */
delta=0x9e3779b9,  /* a key schedule constant */
n=32 ;
while (n-->0) {         /* basic cycle start */
    sum += delta ;
    y += ((z<<4)+k[0]) ^ (z+sum) ^ ((z>>5)+k[1]) ;
    z += ((y<<4)+k[2]) ^ (y+sum) ^ ((y>>5)+k[3]) ;
        }               /* end cycle */
v[0]=y ; v[1]=z ; }
```

Figure 7: The original TEA algorithm (encoding routine).

and first described in their paper in 1994 [6].

The TEA coding routine in C, as published in the original paper, is depicted in Figure 7. The parameters to the coding routine are ***k*** – the 128-bit encryption key represented as an array of four 32-bit words, and ***v*** – the 64-bit plaintext block, which is an array of two 32-bit words. The algorithm is a symmetric-key, Feistel, product cipher that mixes two orthogonal operations – bitwise XOR and addition.

In the "cycle" part of the algorithm, the XOR and addition operations are visible in two lines of code, where the blocks of plaintext are repeatedly mixed together (similar to the operations described in 2.10.5) and with the key material (see 2.10.2), which is repeatedly changing at the beginning of each cycle by changing the *sum* variable value. The *delta*

parameter is chosen to be the Golden ratio ($(\sqrt{5}-1)/2 \sim 0.618034$) multiplied by 2^{31}. On entry to the decryption routine, *sum* is set to be delta * n.

The algorithm uses 16 or 32 iterations, and achieves overall diffusion after only six iterations. The algorithm is fast and has a small footprint, which makes it useful in situations where processing power and memory resources are limited.

The encryption and decryption routines are interchangeable, that is,

$$D_k(E_k(M)) = E_k(D_k(M)) \tag{2.10}$$

The decryption routine performs the exact operations done by the encryption routine but in reverse order.

The reasons for incorporating TEA in this work are of importance when viewing the Coincard application requirements. First, the algorithm is in the public domain and is royalty-free, and can be incorporated into any commercial product, as opposed to many other ciphers that are otherwise suitable to the Coincard application, but are patented by companies or individuals. Furthermore, when considering the platform on which the Coincard application will be implemented on, it becomes critically important to have the encryption algorithm actually fit within the RAM and ROM limitations of the reader application software; this calls for small-footprint, economical algorithms to be used, and again TEA is a natural selection under these conditions.

It is worthwhile noting that weaknesses were pointed out in TEA and thus it was revised in 1997 and again in 1998 (related-key cryptanalysis – [7], [8] and [42]). The weaknesses are largely of theoretical importance only; nonetheless the revised version of TEA is currently believed to have eliminated those weaknesses.

2.12 System security

When considering the overall security of an application such as a Coincard system, the following outline can be used as a guide:

- The development process – which relates to stages of software development, hardware development, and product integration and testing
- The production process – which relates to the actual production of the various parts of the application, such as cards, readers, servers, etc.
- The usage of the system, by its clients.

The protected information in the application can be categorized into:

- Information held within cards, such as their serial numbers or their cash value.
- System information – relates to any cryptographic algorithms, secret keys, and other security methods employed in the system.

The overall system security is always determined by its weakest element. The goals of a good security concept can be summarized in the following list.

- Prevention of physical tampering with security-critical parts.
- Any sensitive functionality is protected from cloning.
- Continued correct operation of security functions must be assured.
- The system must not contain flaws in design, implementation or operation.
- Hardware security mechanisms should be protected against unauthorized disclosure.

- Sensitive information stored in memories is protected against unauthorized disclosure.
- Sensitive information stored in memories should be protected against corruption or unauthorized modification.

It is obvious that the manufacturing phase of any system parts should also be conducted in a secure manner, where sensitive material is always handled appropriately, transferred through secure couriers, etc.

A comprehensive security evaluation criteria is given in [43].

2.13 Hardware security

An important issue that relates to the context of this work is hardware security. Cryptography is one element in the broader context of security. Cryptographic and other security methods are eventually implemented by developers, in a development environment, and realized through hardware, or software which is at some point executed on some hardware, or a combination of both hardware and software.

Any type of card system application would be implemented in silicon, and card readers would be used in a range of applications, in thousands of locations accessible to users.

Attacks against a system's cryptographic equipment can come from three different sources:

1. Outsiders
2. Knowledgeable insiders
3. Funded organizations

While outsiders might only have limited knowledge of some parts of the system, they can take advantage of security weaknesses. Knowledgeable individuals are familiar with the system and might have various degrees of knowledge about its parts and potential access to some of them. Organizations can assemble teams of professionals, perform a comprehensive analysis of the system, and stage sophisticated attacks against it.

The four major attacks against microcontrollers are

1. Microprobing – the chip surface is subject to direct access, such that IC activity can be observed, manipulated, and interfered with.
2. Software attacks – standard communication methods with the IC are used to exploit defects in the communication protocols or weaknesses in their related cryptographic algorithms.
3. Eavesdropping – analog characteristics of the IC's power supply and I/O ports can be accurately monitored during normal IC operation, and information about the IC's inner workings can be deduced.
4. Fault generation – by violating the IC's environmental specifications, for example, supply voltage, clock signal glitches, etc.

A comprehensive description of a range of possible attacks against IC's used in cryptographic systems and smart cards is given in [44] and [45]. The Coincard is not a smart card – but the Coincard reader is essentially software running on a standard microcontroller or a custom-designed ASIC. It is therefore also vulnerable to the same range of invasive and non-invasive attacks mentioned in those papers.

3 ERROR CORRECTING CODES OVERVIEW

3.1 Introduction

An integral part of today's digital communication systems and computers is the use of coding for error control. In order to achieve reliable digital transmission and storage, data is encoded before transmission, in such a way that an error can be detected and/or corrected reliably upon retrieval at the receiving end. In this chapter, the foundations of digital communications are first presented; the concepts of entropy, information, and a communication channel are explained. Section 3.5 formally defines the concept of coding, and subsequent sections focus on the Binary Asymmetric Channel and error correcting codes applicable to it. In section 3.8.3, several upper bounds on the size of asymmetric error correcting codes are presented. Finally, examples for error correcting codes for the Binary Asymmetric Channel are given in sections 3.9 (codes correcting single errors) and 3.10 (codes correcting multiple errors).

3.2 Terminology and concepts

3.2.1 Probability and discrete memoryless sources

An ensemble is a random variable S with a set of possible outcomes

$$S = \{s_0, s_1, ..., s_{K-1}\}, \tag{3.1}$$

having probabilities

$$\begin{aligned} &P(S = s_k) = p_k, \\ &p_k \geq 0, \quad k = 0,1,...,K-1 \end{aligned} \tag{3.2}$$

It is clear that the sum of all outcome probabilities equals 1:

$$\sum_{x \in S} p(S = x) = 1 \tag{3.3}$$

An information source that produces a symbol from the set S once every time unit, and whose symbols are statistically independent, is called a *discrete memoryless source* [16].

3.2.2 The binary channel

Many communication systems are well modeled by the binary symmetric channel (BSC), and the subject of error correcting codes for this channel has been studied extensively. In a binary symmetric channel, the probabilities of a 0→1 or a 1→0 error type are approximately the same; see section 3.4 for a discussion of the BSC properties.

3.3 Information theory basics

3.3.1 Entropy and the binary symmetric channel

A *discrete memoryless information channel* [16]) is a statistical model, specified by the following parameters:

- An input alphabet $\mathrm{X} = \{x_0, x_1, ..., x_{J-1}\}$,
- An output alphabet $\mathrm{Y} = \{y_0, y_1, ..., y_{K-1}\}$,
- A set of transition probabilities $p(y_k | x_j) = P(Y = y_k | X = x_j)$, for all j and k.

Every unit of time, the channel accepts an input symbol $x \in \mathrm{X}$, and in response produces an output symbol $y \in \mathrm{Y}$. The output of the channel is a "noisy" version of its input. The channel is *memoryless,* that is, the output produced at every time unit is dependent only on the input at the same time unit, and not any previous input symbols.

Entropy is a mathematical formulation of the information content or "uncertainty" in a data set. Considering a discrete random variable S defined as above, the *amount of information* gained after observing an event $S = s_k$ is defined [16] as

$$I(s_k) = \log_2\left(\frac{1}{p_k}\right) \tag{3.4}$$

For an event s with probability $p = 1$, it follows that $I(s) = 0$. This means that for an event that is certain, no new information is gained at all. Also, $I(s_j) > I(s_k)$ for $p_j < p_k$. This means that the less probable an event is, the more information is gained by its occurrence.

$I(s_k)$ is thus a discrete random variable with a fixed set of possible values $I(s_0),...,I(s_{K-1})$ with respective probabilities $p_0,...,p_{K-1}$. From the definition of this variable, the notion of *entropy* is derived.

Entropy [16] is the expected value, or average value of the random variable I over the source alphabet, defined as

$$H(\mathrm{S}) = E(I(s_k)) = \sum_{k=0}^{K-1} p_k I(s_k) = \sum_{k=0}^{K-1} p_k \log_2\left(\frac{1}{p_k}\right) \tag{3.5}$$

$H(\mathrm{S})$ is the *average information gain per source symbol* of the variable S.

The joint entropy of the two-dimensional variable (X,Y), characterized by the common probability distribution $p(X,Y)$ is thus

$$H(X,Y) = \sum_{(x,y)\in(X,Y)} p(x,y)\log_2\left(\frac{1}{p(x,y)}\right) \tag{3.6}$$

For two independent random variables X and Y, their joint entropy is additive:

$$P(x,y) = P(x)P(y) \Rightarrow H(X,Y) = H(X) + H(Y) \tag{3.7}$$

The *conditional entropy* of a random variable X given the event $Y = b_k$ took place is the entropy of the probability distribution $P(x|Y = y_k)$:

$$H(X|Y = y_k) = \sum_{j=0}^{J-1} p(x_j|y_k) \log_2 \left(\frac{1}{p(x_j|y_k)} \right) \tag{3.8}$$

It follows from this definition, using the definition of H as a random variable's expectance, that the *conditional entropy of X given Y* is

$$H(X|Y) = \sum_{j=0}^{J-1} H(X|Y = y_k) p(y_k) \tag{3.9}$$

$$= \sum_{k=0}^{K-1} \sum_{j=0}^{J-1} p(x_j|y_k) p(y_k) \log_2 \left(\frac{1}{p(x_j|y_k)} \right) \tag{3.10}$$

$$= \sum_{k=0}^{K-1} \sum_{j=0}^{J-1} p(x_j, y_k) \log_2 \left(\frac{1}{p(x_j|y_k)} \right) \tag{3.11}$$

which is the average of $H(X|Y = y_k)$, summed over all values of y and weighted by their probability; or in other words – the average uncertainty which remains about the variable X after the value of Y is known.

Now, observing the input and output of a binary symmetric channel as two random variables X and Y, if $H(X)$ measures the amount of uncertainty regarding a channel input X, $H(X|Y)$ states the amount of uncertainty remaining about X after observing the channel output Y. If Y gives a perfect reconstruction of the input X, then $H(X|Y) = 0$.

This means there is no uncertainty about X once Y is given, and the channel is then called *lossless*.

From this follows the following definition.

3.3.2 Mutual information

Mutual information is defined [16] as

$$I(X;Y) = H(X) - H(X|Y) \tag{3.12}$$

and following the definition of H,

$$I(X;Y) = \sum_{k=0}^{K-1} \sum_{j=0}^{J-1} p(x_j, y_k) \log_2 \left(\frac{p(x_j, y_k)}{p(x_j) p(y_k)} \right) \tag{3.13}$$

The mutual information is thus the average reduction in the uncertainty about X after observing the channel output Y, or the average information conveyed by Y about X. By symmetry,

$$I(X;Y) = H(Y) - H(Y|X) \tag{3.14}$$

which means that Y tells us about X as much as X tells us about Y, or in formal notation,

$$I(X;Y) = I(Y,X) \tag{3.15}$$

3.3.3 Channel capacity

The channel capacity C of an information channel [16] is the greatest possible mutual information (for any input).

$$C = \max_{\{p(x_j)\}} I(X,Y) \tag{3.16}$$

The capacity C is a function of the conditional probabilities $p(y_k|x_j)$, which characterizes the channel. The capacity C is thus the maximum average mutual information possible over all choices of input probability distributions. It is measured in units of *bits per channel use.*

3.4 The Binary Symmetric Channel

The Binary Symmetric Channel (BSC) is a special case of the discrete memoryless information channel discussed above, with an identical input and output set $x, y \in \{0,1\}$, and with an equal transition probability; i.e., the probability p of a "0" transmission being received as "1" is equal to the probability of a "1" transmission being received as a "0". The following transition diagram represents this channel:

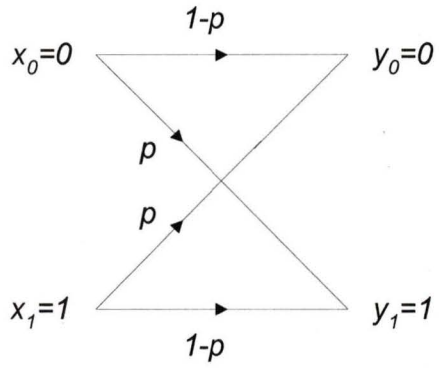


Figure 8: Binary symmetric channel transition diagram.

From the transition diagram, it follows that

$$p(y_0|x_1) = p(y_1|x_0) = p \tag{3.17}$$

and

$$p(y_0|x_0) = p(y_1|x_1) = 1 - p \tag{3.18}$$

The capacity of the BSC can be explicitly calculated; by symmetry, the maximal capacity is achieved when $p(x_0) = p(x_1) = 1/2$. Following the mutual information equation, and substituting values for p and $1-p$ in the entropy equation,

$$\begin{aligned} C &= \max_{p(x_0),p(x_1)} I(X;Y) = H(X) - H(X,Y)\big|_{p(x_0)=p(x_1)=1/2} \\ &= \sum_{k=0}^{1}\sum_{j=0}^{1} p(x_j)\log_2\left(\frac{1}{p(x_j)}\right) - \sum_{k=0}^{1}\sum_{j=0}^{1} p(x_j,y_k)\log_2\left(\frac{1}{p(x_j|y_k)}\right) \\ &= 1 - \left[(1-p)\log_2\frac{1}{1-p} + p\log_2\frac{1}{p}\right] \\ &= 1 + (1-p)\log_2(1-p) + p\log_2 p \end{aligned} \tag{3.19}$$

It is clear that for a channel where $p = 0.5$, $C = 1 + \log_2\frac{1}{2} = 0$ - that is, Y adds no information about X, and no information at all is therefore recoverable from the channel.

A plot of the BSC capacity as a function of p is depicted in Figure 11 (page 56). A comprehensive discussion of channel capacity and the binary symmetric channel is given in [39].

3.5 Introduction to coding

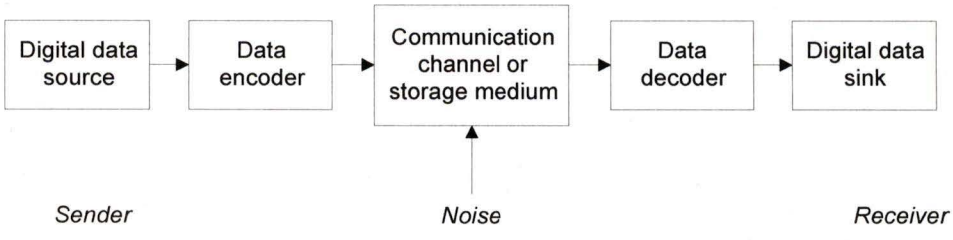


Figure 9: A simplistic model of a digital communication system with coding.

An *Encoder* (see Figure 9) transforms digital information from the digital data source into a coded sequence. There are two different types of coding schemes in use: *block codes* and *convolutional codes* (see 3.5.4).

The simplest form of coding is block coding. In block coding, every data (source) word of length k bits is converted to a *codeword* of length $n = m + k$ bits for transmission through the channel. The *code rate* of an encoder, denoted by R, is the number of information bits entering the encoder per transmitted symbol. For example, for a message of length k, there are 2^k possible messages; if an encoder transforms each k-bit message into a unique n-bit codeword, then the *code rate* (or *information rate*) would be

$$R = \frac{k}{n} \tag{3.20}$$

In order for a code to be useful, it must map each input message to a unique codeword. This dictates that $n \geq k$, or $R \leq 1$. If $n > k$, then m redundant bits can be added to each codeword, thereby enabling error detection and/or correction after the codeword went through the noisy channel. The major issue in designing encoders is choosing the number of redundant bits to add, and what values to assign to them.

3.5.1 Shannon's Fundamental Theorem

Shannon's fundamental theorem [16] states that error-correction trades off reliability (the risk of an uncorrectable error, P_{err}) with information rate R.

Theorem: For any binary symmetric channel with capacity $C > 0$, and any (small nonzero) numbers $\delta, c > 0$ there exists a block code K where

$$P_{err}(K) < \delta \tag{3.21}$$

and

$$R(K) > C - c \tag{3.22}$$

Shannon's theorem thus states that it is possible to find a code as close as desired to both making use of the full capacity of the channel *and* correcting all errors.

3.5.2 Block code properties

The most important characteristic of a block code is its *hamming distance.* The hamming distance $d(a,b)$ between two codewords $a = a_1 a_2 ... a_n$, $b = b_1 b_2 ... b_n$ is the number of bits in which they differ. From this definition, it follows that

$$d(a,a) = 0 \tag{3.23}$$

$$d(b,a) = d(a,b) \tag{3.24}$$

And with c being a third codeword, $c = c_1 c_2 ... c_n$,

$$d(a,b) + d(b,c) \geq d(a,c) \tag{3.25}$$

A block code K is said to *detect* t errors if for each codeword c, all the words derived from c by altering l symbols, where $1 \leq l \leq t$, are not valid codewords.

Examples of block coding include repetition code, parity, CRC, and BCH codes. Decoding of the received codewords can be simply described as comparing each code word against all known (legal) codewords and picking the one most similar.

3.5.3 Linear block codes

Linear block codes are an important subset of block codes. Before defining linear blocks codes, the definitions of a field, and the binary field GF(2) (Galois field) will be given.

Definition A field F is a set $\{0, F'\}$ such that

1. F Forms a group under an addition operation " + ", with 0 being the identity.
2. F' Forms a group under a multiplication operation " * ", with the multiplication result of any element by 0 is 0.
3. The operations satisfy the distributive rule, i.e.,

$$(a+b)*c = a*c + b*c \tag{3.26}$$

Definition GF(2) is the field over the alphabet {0,1} and with the two operations *addition* (+)and *multiplication (*)* defined as follows:

A	B	A+B	A*B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The formal definition [38] of a linear block code is:

- A block code of length n and size 2^k is called a *linear block code* if and only if its 2^k codewords form a k-dimensional subspace of all length-n vectors over the field GF(2).

An important property [38] of linear block codes is that for every two codewords $a, b \in C$, the codeword $a+b$ is also a codeword in C.

In block codes, data is processed in block units, and the encoder has no "memory" – it processes each block separately.

The *hamming weight* (or simply, weight) of a word is the number of its nonzero elements:

$$w(a) = \#\{i | a_i = 1\} \tag{3.27}$$

where the "#" notation stands for "number of".

From the definition of the hamming distance and of modulus-2 addition, it can be shown ([38], pp. 63) that

$$d(a,b) = w(a+b) \tag{3.28}$$

The *minimum distance* $d_{\min}$ of a code K is defined as

$$d_{\min} = \min\{d(a,b) | a,b \in K, a \neq b\} \tag{3.29}$$

A *systematic* code is a code that is composed of two distinct parts – the original message part, and a checking (or *parity*) part, which contains the redundancy necessary for error correction.

3.5.4 Convolutional codes

Convolutional codes were first introduced by Elias [37]. In convolutional codes, the output of the decoder depends not only on the current input block processed, but on m previous input blocks as well. Efficient decoding algorithms exist for this class of codes (majority-logic decoding, sequential decoding, etc.).

A comprehensive survey of block codes and convolutional codes is given in [38].

3.6 The Binary Asymmetric Channel

In some communication systems, the probability of a specific error direction, for example $1 \rightarrow 0$, is much larger than the probability of an $0 \rightarrow 1$ error. This asymmetric error pattern can arise in data transmission systems, as well as in magnetic mass storage media, and in VLSI devices such as memory arrays. For the extreme case, where one error direction has probability zero or near zero, it is common to model the binary asymmetric channel by the *Z channel* (see Figure 10). The $0 \rightarrow 1$ error probability is thus ignored.

Obviously, any error correcting and/or detecting code developed for the symmetric channel will also work in an asymmetric channel. However, code rates may be significantly higher for some asymmetric channel error-detecting codes than the rates obtained by the corresponding symmetric channel codes.

3.7 The Z channel

The Z channel [22] is the digital communication channel with $\{0,1\}$ as the input and output alphabet, and where an error transition $1 \rightarrow 0$ occurs with some positive probability p, and an error transition $0 \rightarrow 1$ never occurs. By exchanging the roles of 0 and 1, the complementary Z-channel is defined. Any code developed for the Z channel will have the same properties on the complementary Z channel if complemented. It can be shown that error-correcting codes for the Z channel have the same properties for the complementary Z channel even without complementation [23].

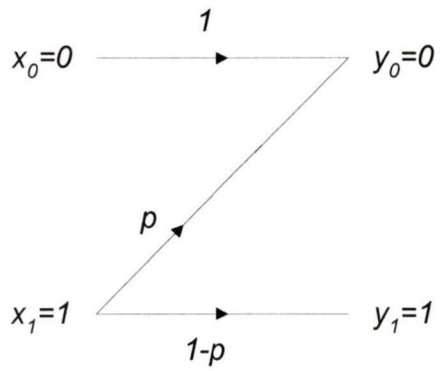


Figure 10: The Z channel.

It is clear from the diagram that the maximal capacity of the Z channel is achieved when all the input symbols are $x_0 = 0$, and no error ever occurs. However, considering the maximal capacity of the channel for an equiprobable distribution of input symbols

$$p(x_0) = p(x_1) = \frac{1}{2} \tag{3.30}$$

the channel capacity can be calculated to be

$$\begin{aligned}
C &= \max_{p(x_0),p(x_1)} I(X;Y) = \{H(X) - H(X,Y)\}_{p(x_0)=p(x_1)=0.5} \\
&= \sum_{k=0}^{1}\sum_{j=0}^{1} p(x_j)\log_2\left(\frac{1}{p(x_j)}\right) - \sum_{k=0}^{1}\sum_{j=0}^{1} p(x_j,y_k)\log_2\left(\frac{1}{p(x_j|y_k)}\right) \\
&= 1 - \left[\frac{1}{2}\log_2\frac{1}{1-p/2} + \frac{1}{2}p\log_2\frac{2}{p} + 0 + \frac{1}{2}(1-p)\log_2 1\right] \\
&= 1 - \left[-\frac{1}{2}\log_2(1-p/2) + \frac{1}{2}p(1-\log_2 p)\right] \\
&= 1 - \frac{1}{2}p(1-\log_2 p) + \frac{1}{2}\log_2(1-p/2)
\end{aligned} \tag{3.31}$$

and is depicted in Figure 12.

$$C = 1 - \frac{1}{2} p(1 - \log_2 p) + \frac{1}{2} \log_2 (1 - p/2)$$

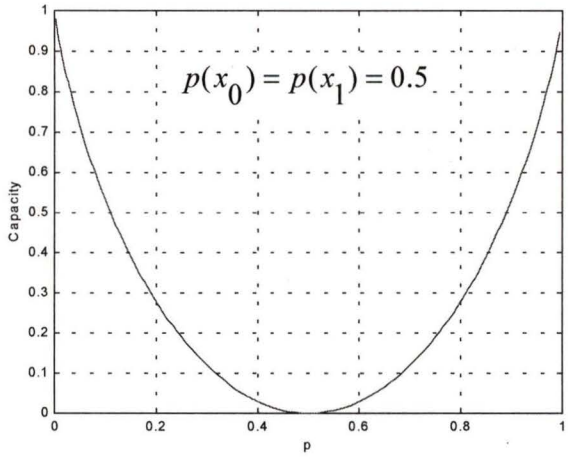


Figure 11: Binary symmetric channel capacity as a function of p

$$C = 1 + (1 - p)\log_2 (1 - p) + p \log_2 p$$

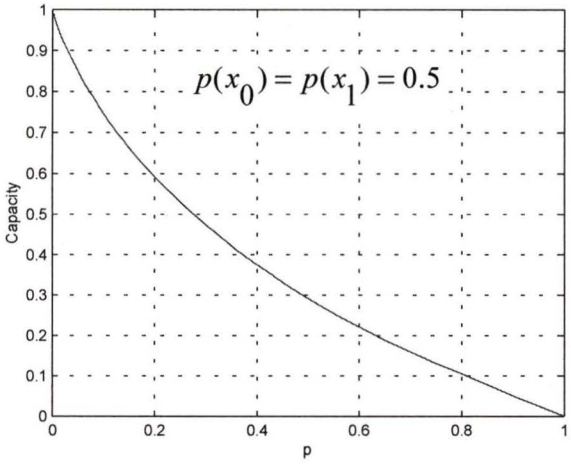


Figure 12: Binary asymmetric channel capacity as a function of p

3.8 Error-correcting codes for the binary asymmetric channel

3.8.1 Definitions

The error patterns in the following discussion can be categorized into three types:

1. A *symmetric* error pattern is a case where both $1 \to 0$ and $0 \to 1$ errors can occur in a single codeword,
2. An *asymmetric* error is an error that can only be of the $1 \to 0$ (*or* the $0 \to 1$) type.
3. A *unidirectional* error refers to a case where $1 \to 0$ and $0 \to 1$ errors are possible, but never occur both in the same codeword.

Let $a = a_1 a_2 ... a_n \in \{0,1\}^n$ and $b = b_1 b_2 ... b_n \in \{0,1\}^n$ be two vectors.

The *number* of coordinates where b is 1 and a is 0 is defined as

$$N(a,b) \equiv \#\{i | a_i = 0, b_i = 1\} \tag{3.32}$$

The *hamming distance* between a and b is $d(a,b) = N(a,b) + N(b,a)$.

- The following relationship is defined between a and b

$$a \subseteq b \tag{3.33}$$

if and only if

$$\forall i, \quad a_i = 1 \Rightarrow b_i = 1 \tag{3.34}$$

- A code C is a collection of binary vectors of a given length n.
- *t-Asymmetric error correcting code (t-code):* A code C is said to be a t-code if it can correct up to t errors.
- The maximal number of codewords in $\{0,1\}^n$ with hamming distance d is denoted by $A(n,d)$. $A(n,d,w)$ denotes the maximal number of vectors in $\{0,1\}^n$ with weight w and hamming distance of at least d apart.

- The maximal size (the total number of codewords) of a t-code of length n is denoted by $\alpha(n,t)$.

- The *asymmetric distance* between a and b is defined as

$$\Delta(a,b) \equiv \max\{N(a,b), N(b,a)\} \tag{3.35}$$

Both d and Δ are metrics on $\{0,1\}^n$. It is clear that

$$w(a) = d(a,0) = \Delta(a,0) \tag{3.36}$$

3.8.2 Basic properties

The necessary and sufficient condition for a code to correct t errors and detect all unidirectional errors (t-EC/AUED) is put forth in the following theorem:

- *Theorem* (due to Kim and Freiman, [23], [20]): a code C is t-EC/AUED if and only if $N(a,b) \geq t+1$ for all $a,b \in C$, $a \neq b$.

In the more general case, the necessary and sufficient conditions for a code to be t-EC/d-UED are outlined in the following theorem.

- *Theorem* [18]: A code C is capable of correcting t unidirectional errors and detecting d unidirectional errors if and only if the following conditions hold For $\forall x, y \in C, \quad x \neq y$,

  - If $N(x,y) = 0$, then $N(y,x) \geq d+t+1$

  - If $0 < N(x,y) \leq t$, then $N(y,x) \geq d+1$

3.8.3 *Upper bounds on* $\alpha(n,t)$

A number of upper bounds are given here for the maximal number of codewords possible for a code of length n bits, correcting up to t errors (denoted by $\alpha(n,t)$). The necessary definitions and the main results will be given, and the proofs are detailed in [23]. Further results on $\alpha(n,t)$ bounds are presented in [31], [32], [34], and [35].

The first bound, due to Varshamov [26], is given as

$$\alpha(n,t) \le \frac{2^{n+1}}{\sum_{j=0}^{t}\left(\binom{\lfloor n/2 \rfloor}{j}+\binom{\lceil n/2 \rceil}{j}\right)}, \tag{3.37}$$

for $n \ge 1$ and $t \ge 1$.

For $t = 1$, we get

$$\alpha(n,1) \le 2^{n+1}/(n+2) \tag{3.38}$$

And for $t = 2$,

$$\alpha(n,2) \le \frac{2^{n+2}}{4+n+\lfloor n/2 \rfloor^2+\lceil n/2 \rceil^2} \tag{3.39}$$

As an example, for the case n=8, $\alpha(8,2) \le 22$. This value is an absolute bound, for any code.

The next bound is given as a solution to an integer programming problem, and is originally due to Goldbaum [24]:

For $n \ge 2t \ge 2$,

$$\alpha(n,t) \le M(n,t) = \max \sum_{r=0}^{n} z_r \tag{3.40}$$

where the maximum is taken over all $z_0,...,z_n$, satisfying the following constraints:

1. z_r are nonnegative integers.

2. $z_0 = z_n = 1$, $z_r = z_{n-r} = 0$ for $1 \le r \le t$.

3. $\sum_{j=0}^{s} \binom{r+j}{r} z_{r+j} + \sum_{i=1}^{t-s} \binom{n-r+1}{n-r} z_{r-i} \le \binom{n}{r}$, for $0 \le s \le t, 0 \le r \le n$.

4. $\sum_{j=s}^{r} A(r-s,2t+2,r-j) z_j \le A(n+r-s,2t+2,r)$ for $0 \le s \le r$.

5. $\sum_{j=s}^{r} A(r-s,2t+2,r-j) z_{n-j} \le A(n+r-s,2t+2,r)$ for $0 \le s \le r$.

This non-explicit bound gives the best upper bound for $\alpha(n,t)$, but it is difficult to compute.

A weaker bound, which is easier to compute, is due to Borden [25]:

For $n \ge t$,

$$\alpha(n,t) \le A(n+t,2t+1) \tag{3.41}$$

also

$$\alpha(n,t) \le (t+1) A(n,2t+1) \tag{3.42}$$

By using the hamming bound, which states

$$A(n,2t+1) \le \frac{2^n}{\sum_{j=0}^{t} \binom{n}{j}} \tag{3.43}$$

It follows that

$$\alpha(n,t) \le \frac{(t+1)2^n}{\sum_{j=0}^{t}\binom{n}{j}} \tag{3.44}$$

For n=8, we get a better bound than that achieved by Eq. 3.39: $\alpha(8,2) \le 20$.

3.9 Codes correcting single errors

There are many construction methods for codes correcting single errors. They include Kim-Freiman codes, Stanley-Yoder codes and Constantin-Rao codes, among others ([19], [22], [36]). Here, a sample construction method due to Delsarte and Piret will be given.

3.9.1 Delsarte-Piret codes

Delsarte-Piret codes [30] are codes of length 7 through 11, with similar construction methods.

Let C_w denote the number of codewords in the code C with weight w.

The construction works by letting $C_w = 0$ for $w = w_1, w_2, ..., w_S$, and using some known combinatorial construction method to get the codewords of weights $w_i + 1, w_i + 2, ..., w_{i+1} - 1$ for $i = 1,2,...$. Constructed that way, it follows that if $w(c_1) < w_i$ and $w(c_2) > w_i$, then $\Delta(c_1, c_2) \ge 2$.

In this construction, the all-zero and all-one codewords $\overline{0}, \overline{1}$ are members of C. It then follows that $C_0 = C_n = 1$ (there is only one codeword of weight 0 and one codeword of weight n), and that $C_1 = C_{n-1} = 0$.

A sample construction of a Delsarte-Piret code of length 11 is given below.

- C_4 and C_7 - are set to 0.

- $C_2 + C_3 = 20$ by using the following construction: it is known that

  $A(12,4,3) = 20$.

  Let X be the code of length 12, constant weight 3, minimum distance 4, and size 20. The set T_{11} is defined as

  $$T_{11} = \{(x_1, x_2, ..., x_{11}) | (x_1, x_2, ..., x_{12}) \in X \quad for \quad x_{12} = 0 \quad or \quad x_{12} = 1\}$$

  by definition it follows that

  $$\forall x, y \in T_{11}, x \neq y, \qquad w(x) = w(y) \quad and \quad d(x, y) \geq 4$$

  or $|w(x) - w(y)| = 1 \quad and \quad d(x, y) \geq 3$

  and in any case $\Delta(x, y) \geq 2$.

  Moreover, it holds that $\forall x \in T_{11}, \quad w(x) \in \{2,3\}$.

- $C_8 + C_9 = 20$ by inverting the elements of T_{11} (all of which are of weight 2 and 3) – thus the set $\{\overline{x} | x \in T_{11}\}$ also holds 20 elements.

- $C_5 + C_6 = 132$ by the following construction: Starting from a (5,6,12) Steiner system[1], and deleting one coordinate, the set R_{11} is defined as

  $$R_{11} = \bigcup_{i=1}^{12} R(i)$$

  where $R(i)$ is the set of cyclic shifts of $r(i)$ defined by

[1] A Steiner system $S(t, k, v)$ is a set X of v points, and a collection of subsets of X of size k (called blocks), such that any t points of X are in exactly one of the blocks.

$$r(1) = (11011100010)$$
$$r(2) = (10110010011)$$
$$r(3) = (01101011010)$$
$$r(4) = (10000111110)$$
$$r(5) = (11110001100)$$
$$r(6) = (11001010101)$$
$$r(i) = \overline{r(i-6)} \quad for \quad 7 \le i \le 12$$

The size of the resulting code is $C_0 + C_{11} + C_2 + C_3 + C_5 + C_6 + C_8 + C_9 = 174$.

For further examples of this construction, the reader is referred to [30] and [23].

3.10 Codes correcting t errors

Many construction methods have been proposed for codes that correct t errors. Various methods are used, including the addition of an error-correcting "tail" of bits, purging (culling) 1-codes of larger length, and generating functions - works by Ray-Chaudhuri [18], Tao [27], Yang [28], Weber ([29], [34]), Zhang [33], and others.

As an example, the construction of Modified Kim-Freiman [21] codes is given in the following section.

3.10.1 Modified Kim-Freiman codes

Let H be a code of length m having hamming distance of at least $2t+1$, and $\overline{0} \in H$.

The t-error correcting code C is defined as

$$C = \left\{ \left(x \middle| x \oplus h_1 \middle| \cdots \middle| x \oplus h_t \right) \middle| x \in \{0,1\}^m \quad , \quad w(x) \quad even, \quad h_1,...,h_t \in H \right\}$$
$$\bigcup \left\{ \left(x \middle| x \middle| \cdots \middle| x \right) \middle| x \in \{0,1\}^m \quad , \quad w(x) \quad odd \right\} \tag{3.45}$$

Choosing H to be as large as possible, the size of the code C would be

$$\#C = 2^{m-1}\left(A\left(m,2t+1\right)+1\right) \tag{3.46}$$

4 SOLUTION DEVELOPMENT

4.1 Introduction

The first step in the solution derivation is the definition and thorough analysis of the requirements raised by the Coincard application, and the range of constraints imposed on any possible solution. The desired security specifications and the error correction requirements define minimal performance needs that must be met, and no upper limit is imposed on any suggested solution except development time. The solution constraints, however, impose limitations that cannot be compromised. Overall, the solution must significantly outperform the existing system.

A discussion of the solution requirements and constraints is presented. It is followed by a description of the development process, describing how the various application, security and reliability requirements played a role in the shaping of the best solution.

4.2 Solution outline

A solution for the layout of the card is sought after, which will allow for the storage of as much information as possible, with as much security as possible, and with the best obtainable error-correction capabilities. A provision for a secure manufacturing process is also sought after. Any solution algorithms should be economical and simple enough as to be implemented on the existing hardware and software platform.

4.3 Solution requirements

4.3.1 Application requirements

1. The card system must allow for the storage of as much information as possible (at least a billion ($1 \cdot 10^9$) distinct cards, subdivided into hundreds, or thousands, of different vendors).

2. The card layout, regardless of the error correction and security scheme being used, must support as many WORM fuses in its geometry as possible.

3. Cost effectiveness: the information storage solution should not impose excessive cost on the existing system. Therefore it is required that the solution should fit into as much of the existing hardware as possible. These include:

    a) PCB layout, and the proven coupling of the card to the sensor array.

    b) Existing processing power available on the reader.

4. Development time.

4.3.2 Provisions for sufficient security

The card security specification addresses the need for security against fraud committed by individuals, by manipulating a single card, or by a factory that has the ability to manufacture any number of cards. The card security specification states the following:

1. It should be *impractical* for an unauthorized plant to manufacture a series of distinct valid cards.

2. It should be *impossible* for an individual to manipulate a card (i.e. by scratching) in such a way that a valid (either new or used-up) card is transformed into a different valid card.

3. It should be *impossible* for a card from a specific vendor's system to function as a valid card in any other system, either as it is or after a manipulation attempt.

By requirement (1), it is clear that the card security cannot rely on keeping the relevant data storage arrangement on the card secret (data hiding or obscuring). Any plant with an access to a number of cards and a working reader can selectively damage data locations on a card until it no longer reads as valid, and in that way recover all the relevant data locations on the card. However, if manufacturing a single valid card is possible, but involves an unreasonable amount of work (for example, manufacturing a million different cards and testing them one by one for validity) then the system is still secure enough. If the cost of fraudulently producing a single valid card is far greater than the value of the card itself, then sufficient security has been achieved.

Requirement (2) addresses a severe problem of data integrity. When considering the solution, it must be impossible to either reload the card, or change its data in any way that would be equivalent to a reloading. By changing a card's serial number, a used-up card or a $5 card can be transformed to a new card or a $20 card, all depending on the nature of the system where it is used. Requirement (3) also relates to an integrity risk, as it states that if the Coincard system becomes prevalent, manipulating an inexpensive card from one system to function as a more valuable card in another system would be impossible.

The security requirement analysis always assumes that the cards are readable to an adversary (for example, by using a standard reader), and that the silicon environment is

safe. Although silicon chips contents have been revealed before using invasive and non-invasive techniques ([44],[45]), they remain the only solution for performing card validation applications out in the field. Creating an on-line database of all cards and transactions is always an option for enhancing a system's security.

4.3.3 Provisions for sufficient error correction

The card platform has several properties that contribute to card read errors. The known problems are

1. Scratching and cutting – either a random or a carefully positioned scratch can penetrate the card's plastic layer and disconnect a bit or a fuse in the internal metal layer.

2. Bending – by repeatedly bending the card, material fatigue in the internal metal layer can change a bit's impedance, thus making the reading of its value more prone to errors. Bending might also permanently disconnect a fuse or a bit.

3. Aging – the internal metal layer is slowly oxidizing, which results in a slowly increasing impedance (see the three sample card material results in Figure 13) and an increased chance of read errors. This is the least likely contributor to read errors, since aging acts on a time scale of years.

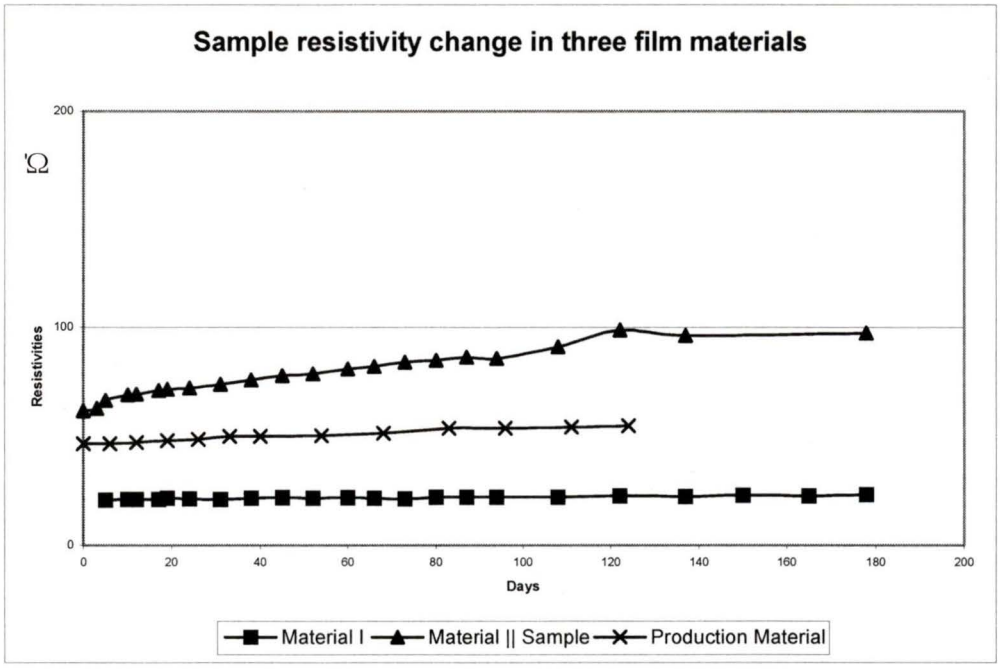


Figure 13: Resistivity changes of three different metal alloy materials as a function of time (source: Coincard International)

Apart from aging, the usual card error pattern is localized; scratches or bends tend to cause a localized error in one or several adjacent bits.

The error correction requirements are summarized in the following list:

1. The error correction scheme must correct *all* 1-bit errors (one damaged bit in the card).
2. The error correction scheme should correct as many errors as possible from the 2-bit, 3-bit errors etc.
3. Error detection is only necessary as to avoid fraud attempts by manipulating a card (this requirement coincides with the security requirement).

Since there is no possibility for data "re-transmission", the function of error detection is thus limited in this application. All that can be done upon an error detection is the declaration of the card as corrupt and unreadable.

4.4 Solution constraints

4.4.1 The tight corner problem

A severe limitation imposed on the card storage capacity is the tight corner problem. The card reader is capacitively coupled to the card's surface, and the measurement performed in order to read a bit's value is essentially a capacitance measurement, where the measured capacitance is on the order of $20pF$. When reading an isolated bit (see Figure 14, left) for example, energy is applied to a pad ***A*** and is passed through the relatively low-impedance fuse to the neighboring pad ***B***, where it is picked up and its strength is measured. However, when considering a read attempt between the same two pads, when the card geometry is populated with many fuses or bits (see Figure 14, right), the "tight corner problem" arises. Energy driving pad ***C*** has several paths to propagate through. Pads ***C*** and ***D*** are interconnected by a 3-bit link, that creates a path that might be low-impedance enough as to make the reading high - as if a "1" bit was present directly between ***C*** and ***D***. The pads are also connected to several other pads, which in turn connect to other pads. The net result is an increase in the crosstalk and stray capacitance, again affecting the reading of the impedance between pads ***C*** and ***D***.

This problem is inherent to the existing card reading hardware and card material characteristics, and it cannot be solved in its current state.

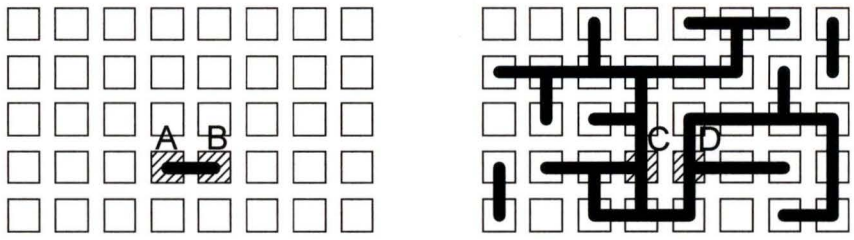


Figure 14: An isolated bit reading (left) vs. a bit reading affected by the "tight corner" problem (right)

The tight corner problem imposes a limitation on the "legal" or valid combinations of "1" bits and WORM fuses that can be placed on a card. Any data layout solution must take this problem into account for the card to be usable.

4.4.2 The writeable fuse problem

A problem similar in nature to the tight-corner problem arises with WORM fuses as well.

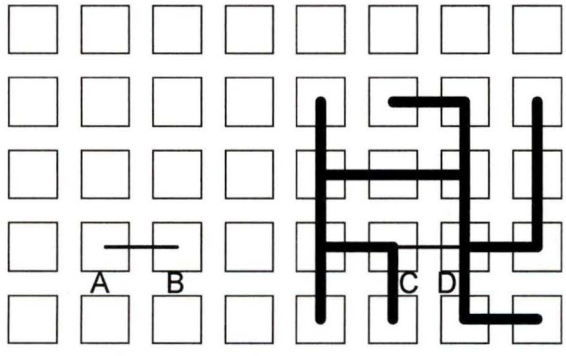


Figure 15: The writeable fuse problem: a fuse that is easily writeable (left) vs. a fuse that might not be possible to electrically disconnect (right)

If a WORM fuse is positioned between pads ***A*** and ***B*** (see Figure 15), most of the energy applied to the pads with the intention to burn and disconnect the fuse will actually be directed to the fuse link itself. However, if a writeable fuse is connected as in between pads ***C*** and ***D***, the energy applied to the pads would dissipate along many different paths, and not all of it will pass through the fuse itself. This leads to situations where a WORM fuse cannot be burned at all, if too many adjacent bits and/or WORM fuses are linked to its pads.

4.5 Solution description

A layout solution that satisfies the error correction and security requirements, and which satisfies the design constraints has been devised. Data layout on the card is composed of 32 "cells" of 2 by 2 pads (4 bits) each. Each pair of cells (4+4 bits) encodes one decimal digit.

The information stored on the card is divided into 10 digits of card information (serial number, vendor ID, etc.) and 6 digits of MAC, an authentication code that authenticates the card information and validates its integrity. The 10 digits of card information are encrypted before they are stored on the card, using a standard 128-bit key encryption algorithm that has been adapted so that it can be used with 32-bit blocks. The value of the 6-digit MAC is also derived using a standard 128-bit encryption algorithm.

The layout offers optional 16 WORM fuse locations. Those locations can be used for other purposes if WORM fuses are not required.

The solution to the security issue is thus implemented through two independent mechanisms: encryption of the card data, and a MAC field on the card that validates the card's data field.

In the following sections, the derivation of the card geometry, error correction, and security aspects of the solution is discussed.

4.6 Card layout

Because of the tight corner problem, a systematic approach to the card data layout must first be devised before an error correction scheme is considered. Once arbitrary data is laid out on the card in some way, the resulting geometry must not have any tight corners, or correct data recovery will be impossible.

One solution to this problem is to reduce the tight corner problem from a global, full card layout problem, to a local layout, or a "symbol" problem. If symbols are geometrically isolated, then the tight corner problem can be solved for the symbol case only, and the global geometric pattern on the card is thus guaranteed to be free from tight corners.

4.6.1 Selection of the 2 by 2 cell

Different sizes and shapes of cells have been considered. A summary of cell geometries and their advantages/disadvantages is given in Table 1.

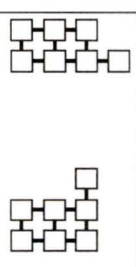
Notes to the table construction method:

- The "*bits per pad*" column states the number of binary bits that fit within a suggested cell geometry. Since pads are a finite card resource, it pays off to use cells with as many bits per pad as possible.
- The table describes the available resources on which an error-correcting code is to be developed: the "*legal raw codewords per digit*" column shows the number of codewords attainable within each digit after removing the all-zero word and all the codewords that violate the tight corner limitation. The "*legal raw codewords*

percentage" column is the percentage from the total (2^n) original codewords left after removing all the illegal ones – a higher percentage means a more efficient representation.

- The number of separate digits available for storage on the card is one half the number of cells that can be arranged on the card. According to error-correction requirement (2), each digit is represented using two far-apart cells on the card.
- The maximal number of WORM fuses was calculated using a simple rule, which states that only one fuse can interconnect two digits, and those two digits cannot be connected to any other digits, since tight corners or very long, meandering paths may result. If a specific geometry always leaves unusable pads, the number of WORM fuses that can be placed over those pads was also added.

Table 1: Cell geometry properties summary.

Cell geometry	Bits per pad	Total number of digits per card	Legal raw codewords per digit	Legal codeword percentage	Maximal number of writeable fuses
	0.5	32	1	50%	-
	0.66	21	3	75%	23
	1	16	11	69%	16
	1	12	23	72%	19
	1.16	10	68	53%	16
	1.14	8	136	53%	~24

It is clear from the table that good bit/pad ratio is achievable with geometries of 4 pads and up. The geometries from 4 through 7 pads would have similar card information capacity, and similar amount of WORM fuses.

The determining factors for the choice of geometry are:

1. Flexibility: Having a large number of digits per card is beneficial. This allows for future data storage requirements to be met by changing the allocation of digit groups for different purposes. With 10 or 8 digits per card, the options for changes in the future are more limited.

2. Codeword space usage: The determining parameter of the code size is the pool of legal, "raw" codewords, available before constraints are applied (for instance, to ensure sufficient hamming distance between different codewords). The percentage of valid "raw" codewords drops significantly for geometries of 6 pads and up.

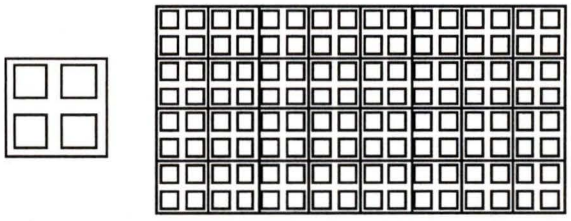
The 4-pad geometry is selected, since it meets the two mentioned criteria, and is also well suited to a straightforward software implementation due to its symmetric allocation pattern, a small error-correction table, and the fact that it leaves no unused pads on the card.

The properties of the 2 by 2 cell layout are summarized below.

1. The 8 by 16 pad card is subdivided into 32 sub-blocks, or "cells", 2 by 2 pads each (see Figure 16).

2. Each cell is an independent data unit, and no data bits connect a cell to its neighboring cells.

3. One WORM fuse is allowed between pairs of adjacent cells, each cell is allowed one fuse attached.



(a) (b)

Figure 16: A single "cell" (a) and the logical cell-based card layout (b)

The sixteen possible fuse layouts for a cell are depicted in Figure 17. Combinations ***11*** through ***15*** are indistinguishable because of the tight corner problem. This means that only one combination can be used from this group, and combination 15 is the suitable one, since it has the most "1" fuses. Combination ***0*** carries too little information, and by disconnecting one or more fuses, all combinations can end up looking like combination ***0***. Therefore the all- "0" geometry is also ruled out. The end result is a set of 11 usable combinations: ***1*** through ***10***, and ***15***.

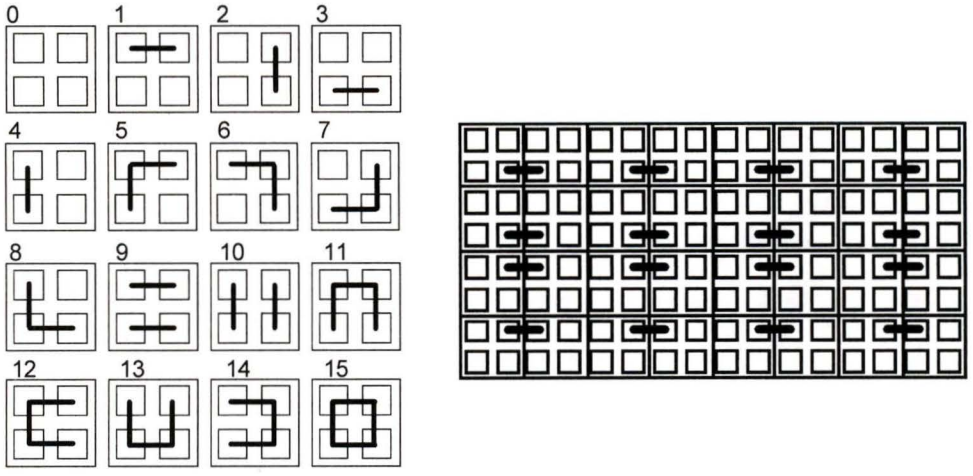


Figure 17: The sixteen possible bit layouts in a 2 by 2 cell (a) and optional WORM fuse locations (b).

In order to satisfy error-correction requirement (2), every application unit of information (hereby referred to as a "digit") is best coded using two physically separate, far-apart locations on the card. This, together with appropriate selection of codewords, ensures that a localized error, where two or more bits are damaged, would not necessarily cause an unrecoverable digit error.

WORM fuses are located as in Figure 17. Note that each fuse has no more than one cell connection on each of its two pads. Thus the likelihood of WORM fuses that cannot be burnt due to too many conduction paths is minimized.

4.6.2 Cell arrangement

In order for the chosen error correcting code to achieve maximum performance, it is best to keep each cell pair of each digit as far apart as possible. This has no positive effect if only

random errors were expected; however, scratches and bends tend to damage a limited portion of the card surface, and rarely two far-apart cells.

A cell arrangement was devised (see Figure 18), where all cells are distributed on the card surface in such a way that cell pairs (marked as ***A*** and ***B***) are positioned far apart. The digits 1 through 10 are the card data field, and the grayed-out digits (11-16) are the card MAC field.

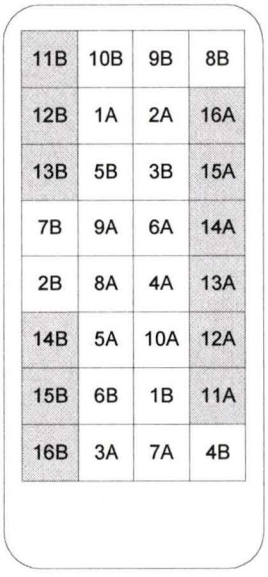


Figure 18: Cell allocation

4.7 Error Correcting Code – introduction

Since the card storage medium is practically a binary asymmetric channel, the only relevant error pattern is in the $1 \rightarrow 0$ direction. Since a "1" reading is always a confident reading, any code being used must include as many "1"s as possible. On the other hand, the security requirements state that a valid card must not be prone to manipulation such that its data may be altered to read as a different valid card.

Considering the coding interpretation of this requirement, it follows that a valid codeword must never be obtained from another codeword by changing "1"s to "0"s. In other words, each codeword used must differ from all other codewords by at least one $0 \rightarrow 1$ transition. By using a coding scheme that follows this rule, it is guaranteed that a random or deliberate scratch can never result in a valid card being turned into a different valid card.

4.7.1 ECC selection

Once the 2 by 2 cell geometry was selected, an error correcting code had to be developed to suit that geometry. Of the 11 valid combinations available within a 2 by 2 cell, some are "weaker" than others. For instance, combinations with only one "1" fuse. In order to minimize the effect of the "weak" codewords, without overly compromising the card data capacity, one of them was dropped (combination 1 in Figure 17), leaving 10 usable codewords. This provides a base-10 representation scheme.

The known Z-channel error correcting code construction methods are unusable in this scenario, since the structure of the desired code is highly irregular –

- It is an "8-bit" binary code, yet many codewords are illegal.
- The size of the code is to be exactly 10.
- The coding of a single digit is to be implemented using two separate – but related - "cells", or groups of 4 bits.

To illustrate these shortcomings, an 8-bit code capable of correcting up to 2 unidirectional errors, given in [29] (Table II), is only of size 6 - and 3 out of those codewords are illegal in the Coincard case. A 9-bit, 10-codeword code described there is also unusable for the same reason.

Because the code size is relatively small, the best error correcting code can be found by simply using an exhaustive search. This procedure is described in the next section.

4.7.2 ECC selection through performance simulation

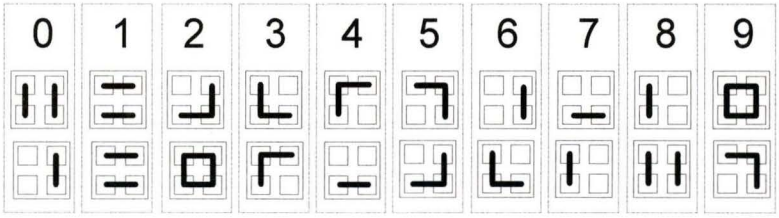


Figure 19: Error correcting code layout sample

Selecting the best two-cell pairing of codewords for error correction capabilities involves checking the 10! (3628800) possible arrangements of the 10 digits in two-cell groups; one such arrangement is depicted in Figure 19. It is intuitively clear that the best arrangement (or arrangements) would be balanced; that is, the number of "1" bits in each cell pair will not be very small (a codeword with only two "1" bits will never be resistant to a 2-bit error) or very large (pairing two four "1" bit words to an eight-bit codeword will create a very robust codeword, but it leaves other codewords with a worse-off error correction capability).

The simulation procedure for selecting the best codes was comprised of the following steps:

1. Each possible code (ten 2-cell pairs) was created using the 10 valid codewords.

2. 16 pseudo-random decimal digits were coded using the ECC generated in step 1, thus a "real" 32-cell card was simulated.
3. The "card" was subject to between 2 and 25 unidirectional ($1 \rightarrow 0$) fuse errors.
4. By using the closest-match method, the most probable value of the 16 erred digits was recovered. Ambiguous matches were treated as unrecoverable errors.
5. An error counter was incremented each time a card was not fully recoverable.
6. For each code, the process was repeated from step 2 for 100 different pseudo-random cards.
7. The code(s) receiving the lowest error counter value will be the best error correcting code(s) for this geometry.

4.7.3 The best ECC candidate

The ECC found (shown in Figure 20) is the best one for the 2 by 2 cell-pair, 10-digit case, for the following reasons:

- Changing any single bit in a single cell from 0 to 1 will either:
    - Create a "tight corner" situation, or
    - Create a cell geometry identical to another cell geometry already in use, thereby causing ambiguity which will always result in less correctable errors.
- Changing any single bit in a single cell from 1 to 0 will:

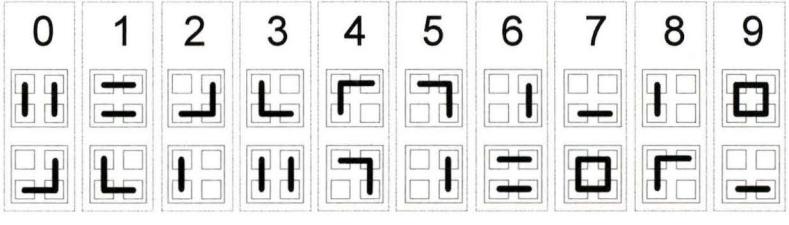
- Always result in a degraded digit representation, able to correct at least one less bit error; any error in the other "1" bits that remain in the digit will be less likely to be correctable without the recently changed bit.
- May result in a codeword (cell pair) that can be changed to another legal codeword by changing a "1" bit to "0", thereby violating the security requirements.

The chosen ECC received the highest mark in the exhaustive search, which took into consideration a full range of error scenarios and code combinations as implemented on a 32-cell card.

The best error correcting code is depicted in Figure 20. The analysis and results of this code's performance are presented in Section 5.2.1.

Index	Cell A	Cell B
0	0101	0110
1	1010	0011
2	0110	0001
3	0011	0101
4	1001	1100
5	1100	0100
6	0100	1010
7	0010	1111
8	0001	1001
9	1111	0010

(a)



(b)

Figure 20: Best ECC binary (a) and graphical (b) representation

4.8 Security solution derivation

It is clear that any security mechanism other than data encryption will not provide adequate security. Hiding or obscuring the data will always be prone to an adversary with access to a large number of cards. Whatever security mechanism is chosen, it is assumed that the security of well-known encryption algorithms is good enough to be used in a suggested solution.

At the core of the security solution is the incorporation of a MAC on the card itself, alongside the card data, which is the "message". The derivation and validation of the MAC are performed only in safe environments (the card production facility or the card reader) and depend on secret quantities known only within those environments. Without knowledge of theses secret quantities, attempts to manufacture a fraudulent card or to manipulate an existing card are rendered either impossible, or uneconomical.

4.8.1 The security solution details

The security solution is depicted in Figure 21 and Figure 22.

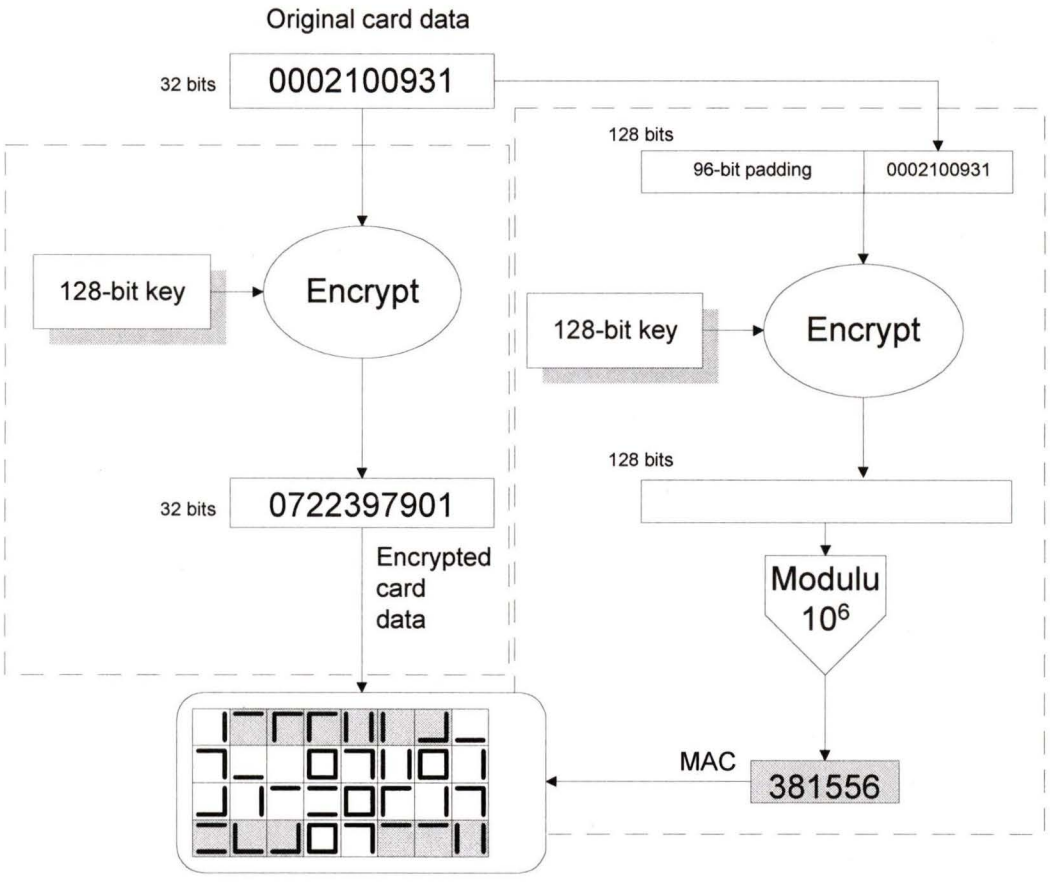


Figure 21: Card manufacturing process flowchart - encryption of card data and creation of MAC.

Figure 21 depicts the process of card production at the manufacturing plant. The production process comprises of two distinct steps: encrypting the card data (left branch in Figure 21), and generating a unique MAC for this card data (right).

The data encryption process is straightforward: a 32-bit word is encrypted and the resulting 32-bit word is placed on the card. Working with 32 bits has the disadvantage of losing a large portion of the available data space: a 32-bit block can represent $4.29 \cdot 10^9$ distinct cards, while the available 10 digits allow for 10^{10} distinct cards. However, this reduction is not critical from the application point of view, and it allows for the usage of a published,

well-studied algorithm; using any unpublished or tailor-made, proprietary algorithms is considered inadequate, since the algorithms and methods have never been analyzed and examined by the scientific community.

The MAC generation process is a little more involved. The 32-bit card data field is first padded to a length of 128 bits using a unique, constant 96-bit padding value. The 96-bit value can be any value, as long as it remains the same for all cards and readers. The 128-bit value is then encrypted, and the result of the encryption is then reduced to a 6 decimal digit value using a simple modulus operation, and the 6 digits are placed on the card.

Figure 22 depicts the card validation and data retrieval process. The card data is first read and an error correction procedure is performed. If error correction has failed, the application is notified. The card data is decrypted using the same algorithm used for encryption. The card's MAC is re-generated using the exact procedure used to generate it in the production process. The re-generated MAC is compared with the MAC read from the card itself. If the two match, then the card is declared valid.

When considering the contribution of the card data encryption scheme, it becomes obvious that the level of security offered by encrypting the data is limited, in the sense that the entire scope of data being encrypted is only a 32-bit plaintext. Irrelevant of the "quality" of encryption algorithm being used, this encryption, being a mapping from 32 to 32 bits, operates on a relatively small plaintext/ciphertext space[1]. Any effort to break the encryption scheme by revealing part or all of the 128-bit encryption key seems unnecessary, since the entire Coincard data scope is only ten decimal digits.

---

[1] Compare with brute force search attacks on a 56-bit key, described in [46] and [47], that are realistic and well within reach of determined attackers.

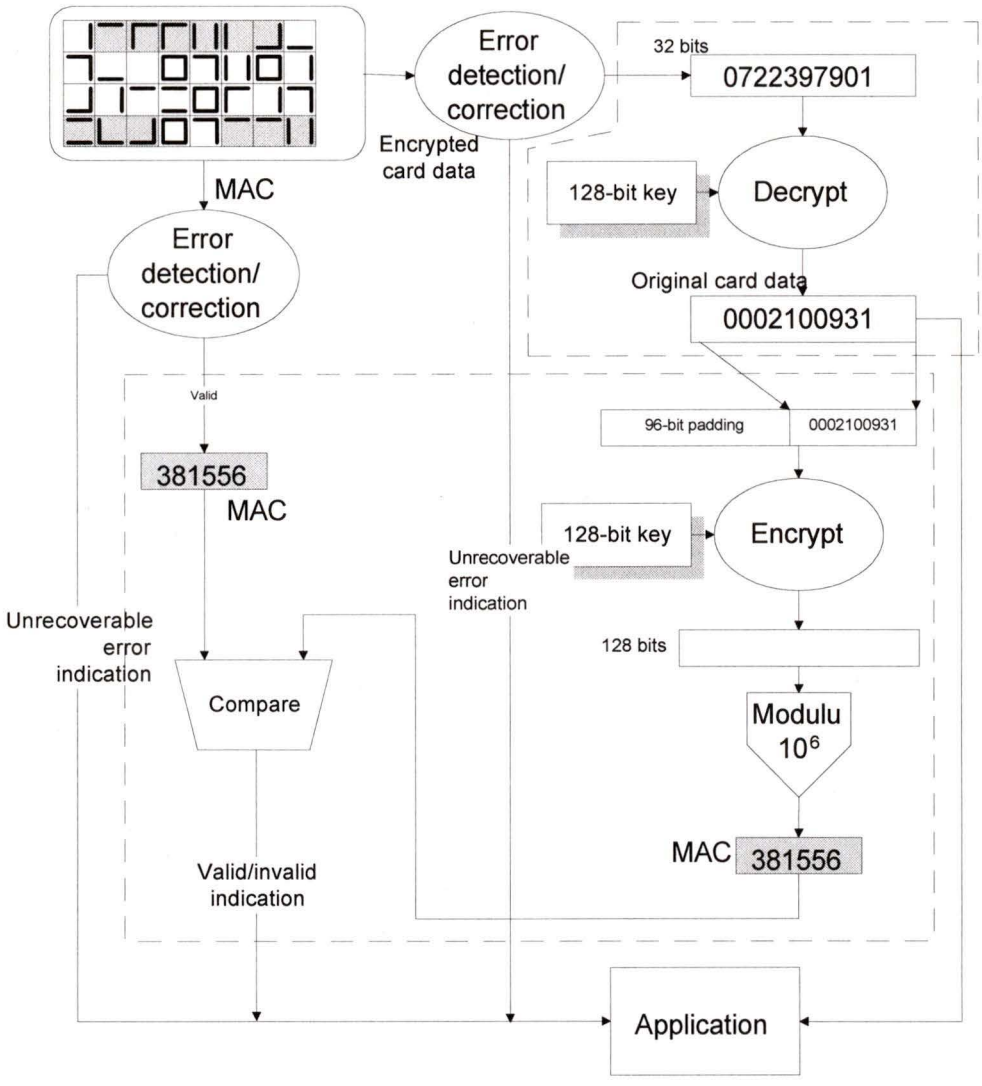


Figure 22: Card verification process flowchart

It is here that the integration of the MAC into the security scheme makes an important contribution – all plaintexts or ciphertexts are "legal" in the sense that they always represent the correct 10-digit information of some legal card, but this fact is not of much use to an attacker since they are useless without the correct MAC, and the MAC derivation process in turn involves a 128-bit secret encryption key and a 96-bit secret padding value,

both remaining embedded inside secure hardware and computationally infeasible to recover.

Having the 10-digit data encrypted on the card costs nothing, yet it serves the important purpose of effectively *hiding* and *obscuring* the card data from being too obvious to figure out. This is true both for an unprofessional fraud attempt done by scratching the card, and for anyone trying to fully understand and break the card data storage scheme.

The cipher chosen to perform the encryption of the data field is TEA [6]-[8]. It is a 128-bit key symmetric cipher, which has been adapted to the requirements of the Coincard application by reducing the block size to 32 bits.

5 SOLUTION ANALYSIS, TESTING AND VERIFICATION

The quality of the solution presented in Chapter 4 is analyzed in this chapter. First, its suitability to the application requirements (outlined in Section 4.3) is discussed. Next, an analysis of its robustness is presented, covering both its error correction capability (Section 5.2) and its cryptographic techniques (Section 5.3). Finally, in Section 5.4, a sample application of a metro transit system is described in the terms of the Coincard solution.

5.1 Application requirements

5.1.1 Information capacity

The information capacity requirement is met, since the card system, as suggested, allows for 2^{32} (4,294,967,296) distinct cards. In case this many cards have already been produced or allocated ("Series I"), a simple procedure exists for increasing the number of valid cards. Changing the secret key used to create the card MAC allows for another set of nearly 2^{32} cards to be manufactured.

If a 6-digit MAC is preserved, then the chances that a card from "Series I" will have the same MAC as a card from "Series II" are one in a million – since the MAC generation process results in a pseudo-random number. Thus, attempting to use a card from "Series I" on a system, which expects a card from "Series II", has a chance of success no greater than 1:1000000.

It is also possible to verify the two MACs are different during production, and skip card serial numbers that accidentally generate the same MAC. An average of 4294 such coincidences is expected for the entire series of 2^{32} cards, and thus the cross-series fraud risk is eliminated completely.

An important feature of the solution layout is its flexibility. It is always possible, if required, to make a card which is more secure but can hold less information by simply increasing the MAC field size (from 6 to 7 or more digits) on the expense of the data field size. It is also feasible to increase the card system information capacity on the expense of security; an 11 or 12-digit data field would still allow for a 4- or 5-digit MAC field to be integrated into the card, which might be enough for a number of applications. All these modifications to the solution require only minor changes and their implementation is straightforward.

An additional advantage of the solution's flexibility is discussed in the following section.

5.1.2 WORM-fuse capacity

The card layout supports 16 WORM fuses as it is. In applications where this number is insufficient, additional fuses can be placed on the card by sacrificing card security and/or information capacity.

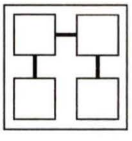


Figure 23: 3 WORM fuses in a cell

By converting a digit cell to WORM fuse storage, 3 additional fuses can be added for each cell (see Figure 23). If, for example, a 3-digit MAC is used instead of a 6-digit MAC, six

cells can be converted to WORM storage. This adds 18 WORM fuses to the existing 16, for a total of 34 fuses. Additional fuses can be added, between existing digits and the newly created WORM fuse cells.

Although this solution has a negative effect on the card's security, still a MAC can take any one of 1000 possible values. Forging such a card, although not impractical, still involves the production of an average of 500 cards before a valid one can be produced.

5.1.3 Cost effectiveness

All the aspects of realizing the proposed solution involve only software changes, i.e., no hardware modifications were required. The execution time for an encryption-decryption process was measured to be approximately 9ms on the existing processor.

Card manipulation

Card manipulation is impossible. Any intentional scratching and bit disconnection always results in one of two possibilities:

- No effect at all - in case the error(s) can be corrected, or
- An invalid card - in case too many bits were damaged within one or more digits.

5.1.4 Cross-vendor fraud

The existing system, using two 128-bit keys, is secure enough as to ensure that a card that was manufactured for a specific vendor will never validate when inserted into another vendor's reader – the distribution of card serial numbers among vendors ensures no card can function in a system other than the one it is intended for. All it takes is for every vendor application to positively identify a valid "Vendor ID" or series number on a card before accepting it.

5.1.5 Provisions for sufficient error correction

The error correction properties are summarized in the following list:

1. The error correction scheme corrects all 1-bit errors in any digit, irrespective of the errors in other digits.
2. The error correction scheme corrects nearly all 2-bit errors in a digit (only 3 out of 41 are uncorrectable),
3. All uncorrectable errors invalidate the card, since there is no way to recover its serial number, and distinguish between valid cards and fraud attempts.

5.2 ECC Analysis

5.2.1 ECC performance

The performance of the chosen ECC was tested using simulation. The idea behind this simulation is to submit a large number of different "cards" to varying degrees of fuse damage, and grade the performance of the ECC for each and every case. While similar to the simulation that was used to select the best ECC itself, the performance analysis simulation evaluates only a single solution over a very large sample space of cards and errors, as opposed to *all* possible solutions over a *limited* sample space of cards and errors.

The ECC was used to encode 50000 pseudo-random 16-digit cards, each of which was subjected to:

1. A range of pseudo-random, unidirectional ($1 \rightarrow 0$) bit errors, between 1 and 23 errors per card.
2. A range of clustered errors; the errors were of the same type and amount as in (1), but with a heavy local characteristic which simulates a localized scratch or bend.

Each pseudo-randomly selected erred bit carried a 50% probability of another bit in an adjacent digit being erred as well; the erred digit and its four immediate neighbors had a 20% chance each of suffering an *additional* error, which in turn carried a 50% chance of an adjacent-digit bit error as well, etc.

The results of this performance simulation are depicted in Figure 24. Since the application requires a full error recovery, the solution performance is measured by its ability to validate an entire card; anything less than full validation is insufficient, since in any application, an invalid card is rejected and thus the system has failed to perform.

Figure 24 shows the performance grade for the six best solutions, which were found in the exhaustive search process. Five solutions gave practically identical results (depicted by the dotted line), while one solution consistently outperformed the others – by up to 4 percentage points. This solution is thus the best ECC suitable for the Coincard application.

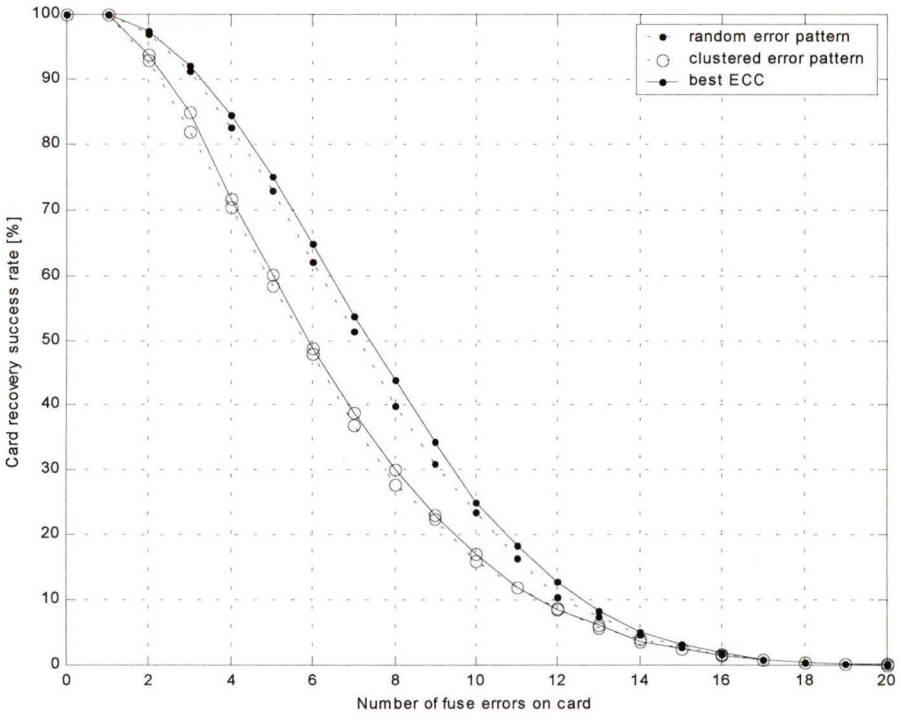


Figure 24: ECC performance simulation results – the best solution's performance is depicted as a solid line

It is clear from the graph that a clustered error pattern leads to poorer performance, since a larger proportion of cells will suffer uncorrectable errors. Overall, the clustered error pattern leads to 10%-20% degradation in card recovery ratio. For pseudo-randomly distributed errors, the recovery ratio is above 85% for 4 errors or less per card, and drops to around 44% for 8 errors per card. The clustered error pattern reaches recovery ratios of 72% and 30%, respectively. The recovery rate diminishes and reaches near-zero levels for 15 errors and up.

It should be noted that the clustered error pattern choice affects the recovery rate results considerably – all pertaining to the "tightness" of the cluster, i.e., the probability for a

number of damaged bits to concentrate in a particular region. A higher tendency for clustering results in lower error correction success rates.

5.3 Security solution analysis

5.3.1 MAC security

The process of deriving the MAC from the card's 10-digit data involves two secret quantities: the unique 96-bit padding value used before encrypting the card's data, and the 128-bit encryption key being used. Since encryption results are pseudo-random in nature, and are completely unintelligible for anyone not knowing the encryption key, it is clear that taking a modulo- 1,000,000 from the 128-bit ciphertext would also result in a pseudo-randomly distributed result. The 6-digit modulo result depends on each on every bit of the 128-bit ciphertext, since it is taken in base 10 (not base 2). There is no way to predict what the ciphertext would be without knowing the key, therefore there is no way to predict the 6-digit MAC for a particular 10-digit plaintext.

5.3.2 Card data encryption

It is worth remembering that a factory capable of manufacturing and testing millions of cards for validity is practically identical to any Coincard authorized factory. It can make thousands of copies of any valid card and distribute them, however this is not a threat, as outlined in the security requirements of the Coincard system.

The only difference, which prevents illicit card manufacturing, is not the inability to manufacture valid cards, but the high cost of producing and testing a series of many thousands of cards before a "valid" one is actually produced. Even with a 10-digit data field, the only economical way of manufacturing valid cards requires knowledge of the

128-bit key. The cost of making an illegitimate card is far higher than the cost of actually buying a genuine card; therefore a sufficient level of security has been achieved.

5.3.3 Attacks against the card system cryptographic methods

Attacks on the card's data encryption mechanism can be any one of the following:

Ciphertext-only attack – since the ciphertext is visible on any card produced.

- Known-plaintext attack - an adversary may have access to an application and a card reader, and can tap communication in some way as to reveal the plaintext (10-digit card data) of a particular card.
- Chosen-ciphertext attack – an unauthorized card manufacturer can produce a card with any desired ciphertext on it, and use a reader to try and validate it.

Chosen-plaintext and adaptive chosen-plaintext attacks are ruled out, since they would require penetrating an authorized Coincard factory and gaining access to the machinery that produces valid cards, or – disassembling a reader and revealing the contents of its hardware, including secret key values. Both those scenarios require access to what is defined in the security requirements as a "safe environment".

Performing any of the three aforementioned attacks is not impossible. The level of security offered by the chosen encryption algorithm is thus the only protection against those attacks. In the described embodiment of this work, the TEA algorithm has been used, and as long as this algorithm is considered safe, the Coincard application is safe as well. In case any weakness is discovered in TEA, which severely affects its security, any other block cipher with similar properties (see, for example, RC6 [9]) can be used.

5.3.4 Attacks against the card system security

The range of attacks applicable against IC's is also applicable to the Coincard reader. The card reader is not a piece of hardware available to every user of the system, but a reader can nonetheless be stolen and analyzed in an attempt to reverse-engineer it. In this case, good design practices of the custom-designed hardware are the protection against invasive or non-invasive attacks. The current implementation of the reader software is on a standard, commercial, code-protected microcontroller platform [41] using the C programming language [40]. On any occasion, the overall card system hardware security can be designed to be at least as good as any alternative system security (such as smart cards, magnetic stripe cards, and their readers).

5.4 Mass-transit fare system sample application

A typical use of the Coincard system as a mass transit fare collection system is described here, with the various parameters explained.

This system, including its secret keys and several types of cards, can be fully defined as below:

"Metro" Transit System Occupies Coincard data range 7000000000 - 7199999999			
128-bit MAC derivation key: 4E0F2C229B37810D967FAF035117E2AE 128-bit data encryption key: 0E24237B2398C12F3F4E03067771A329 96-bit padding: F1887138A206565F93A0CB21			
Card Type	Card Serial Numbers		Remarks
	From	To	
5-Ride Card	7000000000	7049999999	50 million distinct cards; WORM fuses store the number of rides
15-Ride Card	7050000000	7099999999	
Daily Pass	7100000000	7149999999	50 million distinct cards; WORM fuses encode the date
Monthly Pass	7150000000	7199999999	

Coincards can be sold from vending machines, at the company's ticket offices, or on board buses.

Card readers are installed on all buses, ferries and other transport means in the system. They all incorporate a real time clock and calendar, and are connected to a central database using an online- or daily update connection. 5- and 15-ride cards are manufactured and sold with 5 or 15 of their WORM fuses set to "1". Pass- type cards are sold with all 16 WORM fuses set to "1". Upon insertion of a ride-card to a reader, its type is identified according to its 3 uppermost data digits; the WORM fuses are checked, and if the card is

not used up, at least one "1" fuse is located and is burnt. It is also possible to program the card reader to "deduct" more than one fuse from a card if the ride is a "multi-zone" ride etc. by having the driver or the user punch a zone code on a reader's attached keyboard, etc.

When a pass-card is inserted in a reader for the very first time and identified, a different course of action takes place. The 16 available WORM fuses are used to encode the number of days since January 1st, 2000. According to the card's type, the reader writes today's date using those 16 bits, or today's date plus 30 if it is a monthly-pass card. From this point on, whenever this pass card is inserted, a reader compares its date code to "today's" date code; if the date code read from the card is smaller than today's date, it means the card has expired and it will be rejected, giving some kind of audio or visual indication of the rejection.

If the system is connected online to a database, significant security advantages can be achieved. If a ticket or batch of tickets is stolen, it is easy to alert all readers in the system of their serial numbers, and have them rejected. It is also possible to process usage statistics and trace a specific card's usage pattern, and identify fraud attempts performed by an adversary producing multiple copies of a particular card, etc.

Cards that suffered errors due to scratches or bending will go through the error correction process; the more damage inflicted on a card, the more likely it is to be rejected. A card that suffered 6 or 7 destroyed bits has approximately a 50% chance of being validated.

6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

A system solution, which enhances both the security and error resilience of a capacitive memory card, has been presented. In Chapter 1, an overview of the capacitive card technology and its uses has been presented. An overview of cryptography and error correcting codes has been given in Chapters 2 and 3, respectively. In Chapter 4, the system solution requirements were presented, together with the capacitive card's data placement limiting factors. By subdividing the card into small, separate regions (cells), those limitations were resolved. Since an ECC that would not violate the data placement limitations within the cells was difficult to construct, an error correction performance simulation was used to find the best ECC among all the possible ones.

A system security solution that relies on secret keys has been developed and presented. During card production, card data is encrypted and "signed" for authenticity and integrity using a MAC. Upon card reading and verification, the card data is decrypted, and its MAC "signature" is re-calculated and verified against the signature present on the card. A mismatch implies a non-valid card.

In Chapter 5, this solution's performance and quality were assessed. The error correcting code's performance was analyzed using simulation, and all 1-bit errors per card were shown to be correctable, 92% of 2-bit errors were correctable, and a card recovery probability of 50% or more was achieved with as many as 7 bit errors.

The usage of a MAC on a capacitive memory card as a data validation (integrity) mechanism, and as a proof of data authenticity, was shown to be adequate from the security offered, data storage required, and computational overhead aspects. Without prior knowledge of the secret keys being used, manufacturing of a series of valid cards would be impossible or uneconomical to the extent where it imposes no serious security risk. Fraud committed by card manipulation was also rendered impossible. Thus, the inexpensive and disposable card achieves a high security/cost ratio.

A WORM fuse placement scheme that matches the card subdivision without violating the fuse placement limitation has also been presented.

The card manufacturing process security has been addressed (see Section 6.2 below), and drawing from this work's contribution, a safe and controlled manufacturing infrastructure can be designed and employed. The cryptographic methods and data coding mechanisms presented are flexible and general in nature, and can be applied to a range of applications, according to their security, data capacity, and error resilience needs.

6.2 Future work overview

A problem related to the overall Coincard system security is the card-manufacturing problem. Some applications require the manufacturing of millions of cards, possibly using a Laser etching machine situated in some remote location, away from the direct supervision of the authorizing party. It is thus necessary to have some form of control and supervision over the manufacturing process, as to ensure only cards that have been authorized for production have indeed been produced. Section 6.3 discusses one possible solution that will use the contributions of this work to offer a good solution to the secure production

problem. Section 6.4 outlines a suggested solution to a trust problem that might occur between the card system manufacturer and a client, also based on ideas presented in this work. Finally, possible future modifications to the card geometry itself is briefly discussed in section 6.5, modifications that will call for new solutions to new challenges.

6.3 The "Black Box" concept

The "black box" is a hardware device, attached to the Laser production machine by some form of unsecured serial communication. The process of manufacturing a single card involves sending the card's 10-digit data through the communication channel to the black box. The data can then be checked for validity (for example, legal serial number and vendor ID combination), and if valid, the encrypted 10-digit card data and its matching 6-digit MAC will be transmitted back to the Laser machine's controlling computer.

By encapsulating the 128-bit keys and the 96-bit padding value necessary to encrypt the card data and create the MAC inside a secure hardware device (a silicon chip), the keys are never exposed or communicated over any channel during the manufacturing process. Furthermore, the construction of the "black box" is relatively simple, since it performs actions similar to those performed in every card reader.

"Black boxes", being inexpensive pieces of hardware, can be manufactured to suit the needs of any particular manufacturing system. Enhancements such as an Internet-based authorization link between a manufacturing machine's black box and a secure server are also plausible, providing even tighter control over the manufacturing process, while avoiding the weakness associated with communicating encryption keys over a communication system, or storing encryption keys on an insecure manufacturing computer.

A simplistic block diagram of the black box concept is depicted in Figure 25.

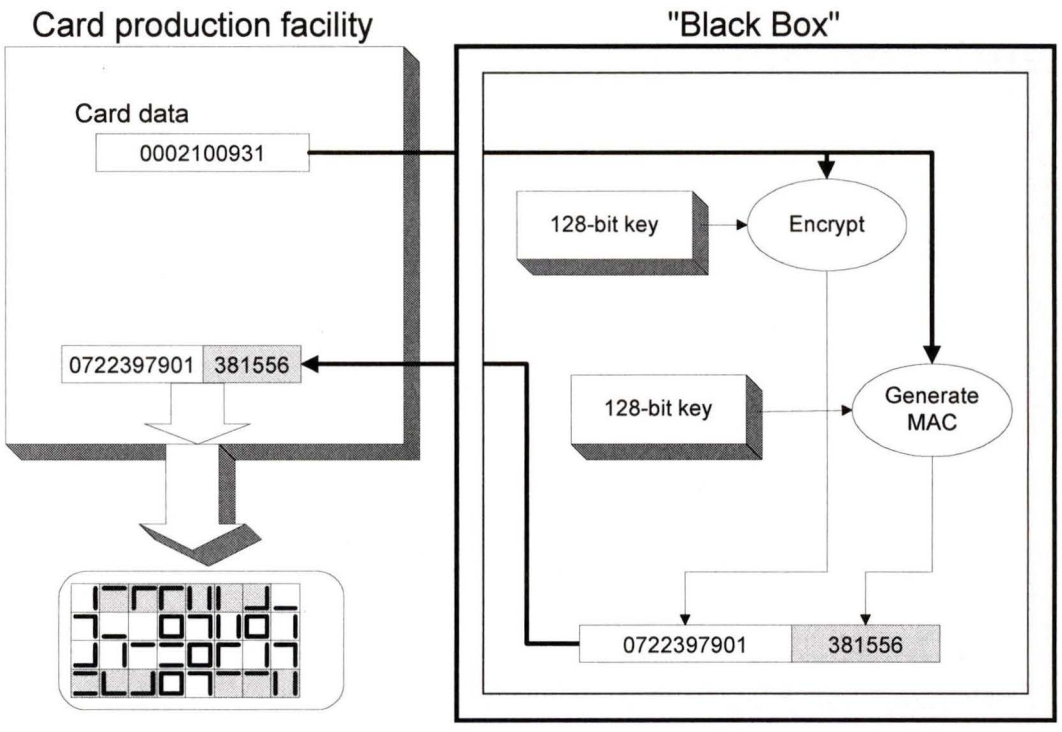


Figure 25: The "Black Box" concept.

6.4 The key sharing problem and suggested solution

The secret keys used in the Coincard application are sensitive information. There are some applications where it is required that neither the card system manufacturer (Coincard), nor the customer, has full knowledge of the keys being used. This implies that trust can be established between the two parties, since no single party can manufacture cards without the knowledge and cooperation of the other party; the customer knows that the system manufacturer cannot produce cards that match his system without his permission, and the card system manufacturer can be assured that the customer cannot exploit his inner knowledge of the Coincard system and its secret keys in breaking the system's security.

By using proprietary hardware, or an embedded microcontroller that allows selective programming and code-protection of specific memory locations, a scheme can be devised

where the two application 128-bit encryption keys are programmed into the black boxes and card readers by both the card system manufacturer and the customer, but are never "shared".

Each party selects two 128-bit keys and keeps their value secret. The card system manufacturer programs its selected keys into their designated PROM locations in the black box and all the card readers to be used in the application, and code-protects those locations such that they are no longer readable from outside the silicon chip. The customer programs his or her selected 128-bit keys in a similar manner to their separate, designated PROM locations and code-protects them.

A suggested method to combine the two 128-bit keys is simply to use one key as a 128-bit key for an encryption algorithm, and the other 128-bit key as the plaintext input to that algorithm. The resulting ciphertext can be used as the application encryption key, since no party can recover it without knowing the other party's key - which is kept secret. This method is graphically described in Figure 26.

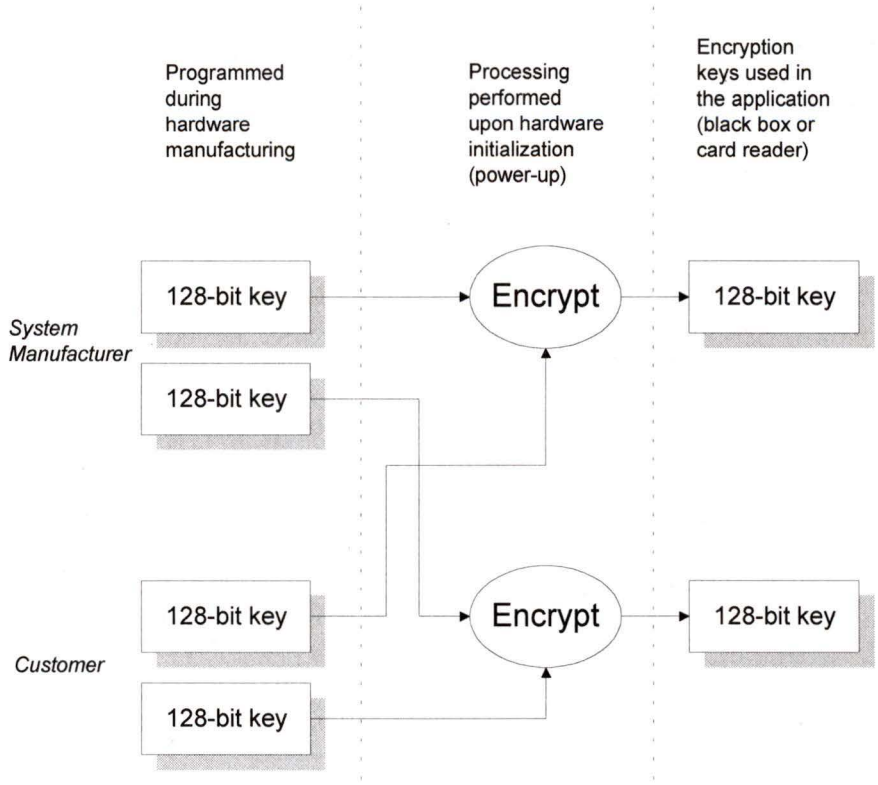


Figure 26: Key-management suggestion schematic operation diagram

The card reader and black box use a secure operation to combine each pair of keys into the final 128-bit key that will be used throughout the system. This key is safe since it does not exist anywhere except during run-time inside the hardware that operated the black box and the reader.

6.5 Alternative geometries

By using a different pad layout, a slightly higher card fuse capacity can be achieved [1]. The usage of hexagonal pad layout (see Figure 27) allows for six bits to be integrated between each hexagon and its neighbors. Although more bits are available for data storage, the tight corner problem still exists but in a more complex version. The development of

this idea would require the development of a new data layout, and a new error correction scheme, which will best match the specific geometry.

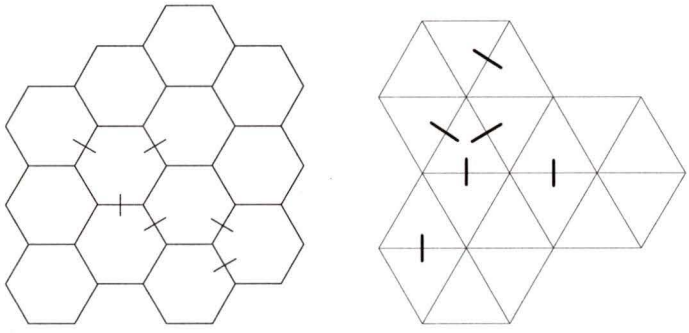


Figure 27: Hexagonal and triangular pad layouts

REFERENCES

1. Schattschneider, et al., "High Security Capacitive Card System," *United States Patent No. 5949060*, Sep. 7, 1999.

2. Machate, Jurgen, "Device for capacitive scanning of a data carrier, marking elements and processes to create the marking elements," *German Patent DE2812388C2*, Oct. 1979.

3. "Identification cards -- Physical characteristics," publication ISO/IEC 7810:1995, and "Identification cards -- Recording technique -- Part 2: Magnetic stripe," publication ISO/IEC 7811-2:1995, *Technical committee / subcommittee: JTC 1/SC 17, International Standards Organization.*

4. Bruce Schneier, "Applied Cryptography," John Wiley & Sons, Inc., Second Edition, 1996.

5. A. Menezes, P. van Oorschot and S Vanstone, "Handbook of Applied Cryptography," CRC press, 1996.

6. David J. Wheeler and Roger M. Needham, "TEA, a Tiny Encryption Algorithm," Computer Laboratory, Cambridge University, England, 1994.

7. David J. Wheeler and Roger M. Needham, "TEA Extensions," Oct. 1997.

8. David J. Wheeler and Roger M. Needham, "Correction to xtea," Computer Laboratory, Cambridge University, England, Oct. 1998.

9. R. Rivest, M. Robshaw, R. Sidney, and Y. Yin, "The RC6™ Block Cipher," *First Advanced Encryption Standard (AES) Conference*, Ventura, CA, 1998.

10. R.C. Merkle, "Secrecy, Authentication and Public Key Systems," UMI Research Press, Ann Arbor, Michigan, 1979.

11. W. Diffie and M.E. Hellman, "New Directions in Cryptography," *IEEE Transactions on Information Theory*, 22 (1976), 644-654.

12. S.M. Matyas, C.H. Meyer and J. Oseas, "Generating Strong One-way Functions with Cryptographic Algorithm," *IBM Technical Disclosure Bulletin,* 27 (1985), 5658-5659.

13. M. E. Hellman and R.C. Merkle, "Public key cryptographic apparatus and method," *United States Patent No. 4218582*, 19 Aug. 1980.

14. H. Feistel, "Cryptography and computer privacy," *Scientific American*, 228 (May 1973), 15-23.

15. D. W. Davies and W.L. Price, "Security for Computer Networks," John Wiley & Sons, New York, 2nd edition, 1989.

16. C.E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal,* 27 (1948) 379–423, 623–656.

17. C.E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal,* 28 (1949), 656-715.

18. D.K. Ray-Chaudhuri, N. M. Singhi, S. Sanyal, and P.S. Subramanian, "Theory and Design of t-Unidirectional Error-Correcting and d-Unidirectional Error-Detecting

Code," *IEEE. Transactions on Computers*, Vol. 43, No. 10, Oct. 1994.

19. S.D. Constantin and T.R.N. Rao, "On the Theory of Binary Asymmetric Error Correcting Codes," *Inform. Contr.* Vol. 40, pp.20-36, 1979.

20. W.H. Kim and C.V. Freiman, "Single Error Correcting Codes for asymmetric binary channels," *IRE. Trans. On Information Theory* IT-5 (1959) pp. 62-66.

21. W.H. Kim and C.V. Freiman, "Multiple Error Correcting Codes for a Binary Asymmetric Channel," *IEEE Trans. on Circuit Theory* CT-6, Special Supplement on International Symposium on Circuit and Information Theory (1959) 71-78.

22. B. Bose and Sulaiman A. Al-Bassam, "On Systematic Single Asymmetric Error-Correcting Codes," *IEEE Transactions on Information Theory*, Vol. 46, No. 2, March 2000.

23. T. Klöve, "Error Correcting Codes for the Asymmetric Channel," Report, Department of Mathematics, University of Bergen, 1981/1983.

24. T. Klöve, "Upper Bounds on Codes Correcting Asymmetric Errors," *IEEE Trans. On Information Theory* IT-27 (1981) 128-131.

25. J.M. Borden, "Bounds and Constructions for Error Correcting/Detecting Codes on the Z-Channel," *Abstracts of Papers, 1981 IEEE International Symposium on Information Theory,* Feb. 9-12, 1981, 94-95.

26. R.R. Varshamov, "Some Features of Linear Codes That Correct Asymmetric Errors," Translated in: *Soviet Physics-Doklady* 10, 1965, 185-187.

27. D.L. Tao, C.R.P. Hartmann and P.K. Lala, "An Efficient Class of Unidirectional Error Detecting/Correcting Codes," *IEEE Transactions on Computers*, Vol. 37, No.

7, July 1988.

28. Yang, C. and Laih, C., "Generating Functions for Asymmetric/Unidirectional Error Correcting and Detecting Codes," *IEICE Trans. Fundamentals,* Vol. E80-A, No.6, June 1997.

29. Weber, J.H, De Vroedt, C. and Boeke, D.E., "Bounds and Constructions for Codes correcting Unidirectional Errors," *IEEE Transactions on Information Theory*, Vol. 35, No. 4, July 1989.

30. Ph. Delsarte and Ph. Piret, "Bounds and constructions for binary asymmetric error-correcting codes," *IEEE Transactions on Information Theory*, IT-27 (1981) 125-128; corrected in IT-36 (1990) 954.

31. Weber, J.H., "Asymptotic Results on Codes for Symmetric, Unidirectional, and Asymmetric Error Control," *IEEE Transactions on Information Theory,* Vol. 40, No. 6, November 1994.

32. F.J.H. Böinck and H.C.A. Van Tilborg, "Constructions and Bounds for Systematic tEC/AUED Codes," *IEEE Transactions on Information Theory,* Vol. 36, No. 6, November 1990.

33. Z. Zhang and C. Tu, "On the Construction of Systematic tEC/AUED codes," *IEEE Transactions on Information Theory*, Vol. 39, No. 5, September 1993.

34. Weber, J.H., De Vroedt, C. and Boeke, D.E., "Bounds and Constructions for Binary Codes of Length Less than 24 and Asymmetric Distance Less than 6," *IEEE Transactions on Information Theory*, Vol. 34, No. 5, September 1988.

35. T. Etzion, "New Lower Bounds for Asymmetric and Unidirectional Codes," *IEEE Transactions on Information Theory,* Vol. 37, No. 6, November 1991.

36. S. Al-Bassam, R. Venkatesan and S. Al-Muhammadi, "New Single Asymmetric Error-Correcting Codes," *IEEE Transactions on Information Theory*, Vol. 43, No. 5, September 1997.

37. P. Elias, "Coding for Noisy Channels," *IRE Conv. Rec.*, Part 4, pp.32-47, 1955.

38. Shu Lin and Daniel J. Costello, "Error Control Coding," Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1983.

39. Haykin, S., "Digital Communications," John Wiley & Sons, Inc. 1988.

40. "PIC C Manual," HI-TECH Software, P.O. Box 103, Alderley, QLD 4051, Australia, fourth printing, February 1999.

41. "PIC17C7XX, High-Performance 8-Bit CMOS EPROM Microcontrollers with 10-bit A/D," Microchip Technology, Inc. 1999.

42. J. Kelsey, B. Schneier and D. Wagner, "Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA," *1997 International Conference on Information and Communications Security*, Beijing.

43. "Common Criteria for Information Technology Security Evaluation Protection Profile - Smartcard Integrated Circuit," a paper issued by Motorola semiconductors, Philips semiconductors, Service Central de la Sécurité des Systèmes d'Information, Siemens AG semiconductors, STMicroelectronics and Texas Instruments semiconductors; Registered at the French Certification Body (Evaluation et Certification Française) under the number PP/9806, Ver. 2.0, Sept.

1998.

44. Anderson, R. J. and Kuhn, M.G., "Low Cost Attacks on Tamper Resistant Devices," *proceedings of the 1997 Security Protocols Workshop*, Paris, April 7-9, 1997.

45. Anderson, R.J. and Kuhn, M.G, "Tamper Resistance --- a Cautionary Note," *The Second USENIX Workshop on Electronic Commerce Proceedings*, Nov. 1996, pp 1—11.

46. M.J. Wiener, "Efficient DES Key Search," TR-244, School of Computer Science, Carleton University, May 1994.

47. F. Hendessi and M.R. Aref, "A successful Attack Against the DES," *Third Canadian Workshop on Information Theory and Applications,* Springer-Verlag, 1994, pp 78-90.

VITA

Surname: Gendel Given Name: Alon

Place of Birth: Kfar-Saba, Israel

Educational Institutions Attended

University of Victoria	1999-2001
University of Tel-Aviv	1992-1996

Degrees Awarded

B.Sc.	University of Tel-Aviv	1996

Publications

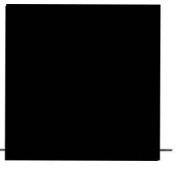
1. "*Frequency hopping communication method and apparatus for modifying a frequency hopping sequence in accordance with counted errors*", US. Patent No. 6115407 (with others), September 2000.
2. "*Automatic transmission power level control method in a frequency hopping communication system*", US. Patent No. 6115408 (with others), September 2000.
3. *"Apparatus for and method of providing an indication of the magnitude of a quantity",* US. Patent No. 6127936 (with others), October 2000.

UNIVERSITY OF VICTORIA PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Title of Thesis/Dissertation:

Reliable and Secure Information Storage on a Capacitive Memory Card

Author 

Alon Gendel

April 25, 2001