ERROR DETECTION AND CORRECTION ON THE CREDIT CARD NUMBER USING LUHN ALGORITHM

LEAH WANGECI WACHIRA

MASTERS OF SCIENCE
(Pure Mathematics)

JOMO KENYATTA UNIVERSITY OF AGRICULTURE AND TECHNOLOGY

2016

Error Detection and Correction on the Credit Card Number using Luhn Algorithm

Leah Wangeci Wachira

A Thesis Submitted in Fulfilment for the Degree of Master of Science in Pure Mathematics in the Jomo Kenyatta University of Agriculture and Technology

2016

DECLARATION

This thesis is my original work and has not been presented for a degree in any other University.

Signature..... Date.....

Leah Wangeci Wachira.

This thesis has been submitted for examination with our approval as University Supervisors.

Signature..... Date.....

Dr. Waweru Kamaku.
JKUAT, Kenya.

Signature..... Date.....

Dr. Lewis Nyaga.
JKUAT, Kenya.

DEDICATION

I dedicate this project to my beloved parents, Mr. Charles Wachira and Mrs. Charity Wachira, brother Benson Wachira and sisters Faith Wachira and Martha Wachira who are source of my inspiration.

ACKNOWLEDGMENTS

First and foremost , I thank the Almighty God for the gift of life and for being with me this far. I am grateful to my supervisors Dr. Waweru and Dr. Nyaga for their guidance, support and mentorship throughout my university education and in shaping my career, to the department of Pure and Applied Mathematics for providing me with all the necessary materials, support and developing me academically and to the Jomo Kenyatta University of Agriculture and Technology fraternity for providing the best environment for learning and developing my career. Special thanks to my friend Vincent Mwai for his support and guidance.

TABLE OF CONTENTS

LIST OF TABLES

SYMBOLS AND ABBREVIATIONS

(a, b)	GCD of a and b
$\forall$	For all
a, b	Code word digits
$a \cdot b$	The product of two digits.
$a \mid b$	a divides b
$a \nmid b$	a does not divide b
ab	A two digit number.
C	Code C
$GF(q)$	Galois f eld of order q
i, j, k	Digits positions in a codeword
IBM	International Business Machines
u, v, x, y	Vectors in a code
$V(n,q)$	Set of all ordered n-tuples over $GF(q)$
w_i, w_{i+1}	Weights in a formula
F_q	Set of q distinct elements
GCD	Greatest Common Divisor
IMEI	International Mobile Equipment Identity

ABSTRACT

The Luhn algorithm is used in credit card number for validation purposes. The algorithm was designed to cater for document storage and retrival in IBM. But later it was improved to cater for error detection purposes in credit cards and IMEI numbers among other applications. The algorithm does not detect jump transposition errors, some twin errors, some adjacent transposition errors and also does not have any error correction capabilities. In this research it is observed modulo 10 is not effective in error detection since it allows errors associated with zero divisors not to be detected. Also, the Luhn algorithm does not detect some twin errors and transposition of digit 09 for 90 due to the reduction of the products to a single digit. Jump transposition errors can only be detected if the weights of the algorithm are changed which ultimately changes the Luhn algorithm. A modulo 13 algorithm is then designed with its error detection capabilities discussed and analyzed and then a convertion tool for existing codes in Luhn algorithm to the new modulo 13 algorithm code is given. It is shown that the modulo 13 algorithm surpasses the Luhn algorithm in as far as error detection is concerned.

CHAPTER ONE
INTRODUCTION AND LITERATURE REVIEW

1.1 Background of the Study

The greatest common divisor (GCD) of the integers $a_1, a_2, a_3, ..., a_n$ is the largest natural number d which divides all a_i denoted as $(a_1, a_2, a_3, ..., a_n)$ or $\gcd(a_1, a_2, a_3, ..., a_n)$. A number $p > 1$ is called a prime number if it is divisible only by ± 1 and by $\pm p$ otherwise it is composite, that is if n is composite then integer factors m and k can be found such that $n = mk$, $m > 1$, $k > 1$. Two integers are said to be relatively prime or coprime if their gcd is 1.

According to Rosen (2012), an algorithm is a finite sequence of precise instructions followed when performing a computation or solving a problem. Algorithms are used for calculations, data processing and automated reasoning. There are several properties that algorithms generally share. These properties are:

- Input: Every algorithm has input from a specified set such as the set of integers or natural numbers among others.
- Output: From each set of input values an algorithm produces an outcome or output values form a specified set.
- Correctness: It is expected that from each set of input values the algorithm should produce the correct output values.
- Definiteness: All the steps of an algorithm should be well defined and precise to avoid confussion.
- Finiteness: After a finite number of steps an algorithm should produce the desired set of output from any given set of input.
- Effectiveness: It must be possible to perform each step of an algorithm exactly and in a finite amount of time.
- Generality: The procedure should be applicable for all problems of the desired form, not just for a particular set of input values.

According to Celko (2010), checksum algorithms are small-sized datum computed from an arbitrary block of digital data for the purpose of detecting errors which may have been introduced during transmission or storage. A check digit is a form of redundancy check used for error detection on identification numbers. It consist of an alphabet or a single digit, sometimes more than one, computed by an algorithm from other digits or letters

in the sequence input. With a check digit one can detect simple errors in the input of a series of characters (usually digits) such as a single mistyped digit or some permutations of two successive digits. Weighted sum refers to multiplication of the elements of a code word by a constant, referred to as the weight, before calculating the checksum.
An error is a deviation from accuracy or correctness which is commonly caused by noisy communication channels such as thermal noise, imperfections in equipments, human errors among others. If a code word u is transmited through a noisy comunication there is a possibility that a different code word v will be received instead of the original code word u hence presence of an error in the code word. Error detection is the identif cation of errors in a code word of which it may be discarded and request for retransmission made. Error correction is the detection of errors in a code word and reconstruction of the original error free data. There are several types of errors that occur during data entry (Weisstein, 2015; Gallian, 1991). Such as:

- Single digit error which occur when a digit is mistaken for another or when a digit is unclear or smudged. This are the most common errors making up to 60% to 90% of the errors made. They are of the form $a \longrightarrow b$ such as $31 \longrightarrow 37$.

- Transposition errors occur when digits are interchanged or reversed. This affects the permutations changing the checksum and the check digit as a result. They are usually of the form $ab \longrightarrow ba$ for example $38 \longrightarrow 83$.

- Jump transposition error occur when three digits are reversed such as $abc \longrightarrow cba$ resulting in a wrong checksum and eventually a wrong check digit. For example $123 \longrightarrow 321$.

- Jump twin errors occur when two equal digits are interchanged for another such as aba becoming cbc which eventually causes change in the checksum and in the check digit. For example $323 \longrightarrow 424$.

- Phonetic errors are made when a digit is replaced with another that sounds almost the same. They are usually in the form of $a0 \longleftrightarrow 1a$ where $2 \leq a \leq 9$. For example $19 \longrightarrow 90$.

- Twin errors are made when a pair of similar digits are replaced with another pair, that is aa replaced with a pair of digits such as bb. For example $44 \longrightarrow 66$.

Identif cation codes are codes that assign numbers or symbols to objects, items or even human beings for easy identif cation. Error-detection identif cation codes are codes that

add extra digits to the identif cation numbers by formulas or algorithms that allow detection of various types of errors such as single errors, transposition errors among others (Sutherland, 1990).
A f eld consists of a set F and two binary operations "+" (addition) and "$\cdot$" (multiplaction), def ned on F, for which the following conditions are satisf ed:

- $(F,+,\cdot)$ is a ring.
- Multiplication is commutative: $\forall a,b \in F, a \cdot b = b \cdot a$.
- Multiplicative identity: $\exists 1 \in F$ such that $a.1 = 1.a = a, \forall a \in F$
- Multiplicative inverse: If $a \in F$ and $a \neq 0$ $\exists b \in F$ such that $a \cdot b = b \cdot a = 1$

According to Raymond, if C is a code then the members in the code are called code words. The elements that make up the code word are referred to as the bit strings. A Galois f eld, $GF(q)$, is a f nite f eld of order q such as the set $\mathbb{Z}_q$ where q is prime. A linear code over $GF(q)$ is a subspace of $V(n,q)$ in some positive integer n. Thus a subset of $V(n,q)$ is a linear code if and only if $u+v \in C, \forall\, u, v \in C$ and $a \cdot u \in C \ \forall\, u \in C$, $a \in GF(q)$ (Raymond, 1986).

Example 1.1. $C = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$ is a linear code of $V(4,2)$

If a and b are integers and m is a positive integer, then a is congruent to b modulo m if $m|(a-b)$ or $a-b=mk$ where $k \in \mathbb{Z}$. The notation $a \equiv b\ (mod\, m)$ is used to indicate that a is congruent to b modulo m. In this case m is said to be the modulus while $a \equiv b(mod\, m)$ is referred to as the congruence (Raymond, 1986). Let p be a prime number. The integers modulo p, consisting of the integers $\{0,1,2,...,p-1\}$ with addition and multiplication performed modulo p, is a f nite f eld of order p. If the modulus p is composite, that means $p = a \cdot b\ \forall\, a, b \in \mathbb{Z}_p$, not all elements in the set of integers $\{0,1,2,...,p-1\}$ have inverses. Therefore a composite modulus is not a f eld due to the presence of zero divisors. For any integer a, a mod p shall denote the unique integer remainder r, $0 \leq r \leq p-1$, obtained upon dividing a by p. This operation is called reduction modulo p. Therefore, f nite f elds are f elds that have a f nite number of elements and this number is called the order of the f eld.

Theorem 1.1. *The ring $\mathbb{Z}_n$ is a f eld if and only if n is prime* (Bowman, 2010, p.48; Raymond, 1986, p.35).

Lemma 1.1. *Any f eld F has the following properties :*

- $a0 = 0, \forall$ *a in F*
- $ab = 0 \Longrightarrow a = 0$ *or* $b = 0$*(thus the product of two non-zero elements of a f eld is also non-zero)* (Raymond, 1986, p.36).

Theorem 1.2. *Suppose an identif cation number* $a_1a_2...a_n$ *satisf es the dot product* $(a_1, a_2, ..., a_n) \cdot (w_1, w_2, ..., w_n) \equiv 0 \, (mod \ k)$. *Then a single-position error* $a_i \longrightarrow a_j$ *is undetectable iff* $(a_i - a_j)w_i \equiv 0 \, (mod \ k)$ *and a transposition error that interchanges the elements in the* i^{th} *and* j^{th}*positions is undetectable iff* $(a_i - a_j)(w_i - w_j) \equiv 0 \, (mod \, k)$ (Gallian, 1991, p. 198).

A q-ary code is a given sequence of symbols, where each symbol is chosen from a set $F_q = \lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$ of q distinct elements. The set F_q is called the alphabet and is often taken to be the set $\mathbb{Z}_q = \{1, 2, 3, ..., q-1\}$. For example the set $F_3 = \mathbb{Z}_3 = \{0, 1, 2\}$. $(F_q)^n$ is the set of all ordered $n-$tuples $a = a_1, a_2, a_3, ..., a_n$ where each $a_i \in F_q$. The elements of $(F_q)^n$ are called vectors or words. The order of the set $(F_q)^n$ is q^n (Raymond, 1986).

Theorem 1.3. *Suppose an error detecting scheme with an even modulus detects all single-position errors. Then for every i and j there is a transposition error involving positions i and j that cannot be detected* (Gallian, 1991, p. 199).

The minimum distance of a code C denoted as $d(C)$ refers to the smallest of the distances between distinct code words. That is $d(C) = min \, \{d(x,y) \mid x, y \in C, x \neq y\}$.
Example 1.2. A code C of $V(2,2)$ has minimum distance of $d(C) = 1$.

$$C = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

The hamming distance between two vectors x and y of $(F_q)^n$ is the number of places in which they differ. It is denoted by $d(x,y)$. Let $x = 01101$ and $y = 11100$ be elements of $(F_2)^5$ the hamming distance between x and y, that is $d(01101, 11100) = 2$. Hamming distance is a distance function since it satisf es the four conditions:

- $d(x,y) \geq 0$
- $d(x,y) = 0$ iff $x = y$.
- $d(x,y) = d(y,x) \forall x, y \in (F_q)^n$

- $d(x,z) \leq d(x,y) + d(y,z)$, $\forall\, x,y,z \in (F_q)^n$ (Raymond, 1986).

The Luhn algorithm or the Luhn formula is a simple checksum formula used to validate a variety of identif cation numbers, such as credit card numbers , IMEI (International Mobile Equipment Identity) numbers, National Provider Identif er numbers in US and Canadian Social Insurance Numbers. This algorithm was created by a scientist called Hans Peter Luhn in 1950*s*. It was designed to protect against accidental errors that commonly occur during data entry. Most credit cards and many government identif cation numbers use the algorithm as a simple method of distinguishing valid numbers from mistyped or otherwise incorrect numbers (Hodge, Schlicker & Sundstrom, 2013). According to Child's (1979), the Luhn formula has no error correction capabilities. The following is an illustration of how the Luhn formula works in validation of account numbers and in calculation of the checkdigit.

1.1.1 Credit card validation using Luhn formula.

The formula verif es a number against its included check digit. Counting from the check digit, which is the rightmost and moving left, double the value of every second digit. If for all the products $p < 10$ where p is equal to either n or $2n$, then calculate $\left(\sum_{i=0}^{k} 2n_{2i} + \sum_{i=0}^{k} n_{2i+1}\right) mod\ 10$, where $n \in \mathbb{Z}_{10}$ and $k \in \mathbb{N}$ but if $\exists\ p \geq 10$ add the digits of the product to obtain a single digit, that is $\forall\, a, b, c \in \mathbb{Z}_{10}$, $p = ab = a + b = c$ before calculation of the checksum *mod* 10. If the total modulo10 is equal to 0 (if the total ends with zero) then the number is valid according to the Luhn formula otherwise it is invalid (Ghandhi, 2015) .

1.1.2 Check digit calculation using Luhn algorithm

Let $a_1a_2...a_n$ be an account number that will have a check digit added making it of the form $a_1a_2...a_nx$. From the rightmost double every even positioned digit to obtain $v_1v_2...v_nx$. If $\exists\, v_i \geq 10$, sum the digits of the product (e.g. $10 = 1+0 = 1, 18 = 1+8 = 9$) to obtain a single digit before computing the check digit $x = 10 - (\sum_{i=1}^{n} v_i)\ mod\ 10$ but if $\forall\ v_i$, $v_i < 10$ calculate the checkdigit x directly as $x = 10 - (\sum_{i=1}^{n} v_i)\ mod\ 10$. The check digit x will only be zero if the checksum is a multiple of 10 or has 10 as a factor otherwise the check digit is in the set $\{1, 2, ..., 9\}$.

Example 1.3. The check digit x of $(888x)$ using the Luhn formula is

Table 1.1: Check Digit Calculation in Luhn Formula

Number under consideration	8	8	8	x
Multiply every even positioned digit from the rightmost by two to get product p	16	8	16	x
If $p \geq 10$, that is $p = ab, \forall\, a, b, n \in \mathbb{Z}_{10}$ add the digits together, $a+b=n$	7	8	7	x

$$checksums = \sum_{i=0}^{4} n_{2i} + \sum_{i=0}^{4} n_{2i+1} = 22.$$

If $\forall\, p,\ p < 10$ calculale the checksum $= \sum_{i=1}^{k} p_i$ where $k,\ i \in \mathbb{N}$.

$$checkdigit\ x = 10 - checksum(s)(mod\ 10) = 8$$

Therefore the number that passes the Luhn formula is 8888 (Ghandhi, 2015) .

1.2 Literature Review

Error correction and detection has been a key area for many researchers with the aim of either creating new codes or enhancing the ones that exist. Tilborg (1993) described a channel as a medium over which information is sent together with its characteristics which consist of an input X , an output Y and a transition probability function P. Henk observed that unless stated otherwise , successive transmissions have the same transition probability function that are independent of each other.

Kumar and Kaur (2015) studied the IMEI number and noted that despite it being used world wide for unique identif cation of mobile equipments hackers have designed various methods to change the IMEI number of mobile devices so that stolen or lost mobile equipments are not located by their owners or security agencies. They concluded by emphasising on a need of an improved security measures that can not be hacked.

Gupta and Verman (2012) stated that cyclic codes are linear block codes that can be obtained by cyclically shifting all the elements of a code word to the left forming a new code word. A code word $a_1a_2a_3\cdots a_{n-1}a_n$ of a cyclic code C when cyclically shifted, code word $a_2a_3\cdots a_{n-1}a_na_1$ is obtained. Cyclic redundancy check or CRC are common form of cyclic codes used for error detection whereby the check bits are transmitted along with the data to the receiver where the receiver performs similar calculations to compare the CRC check bits received.

Ghandhi (2015) analyzed the various types of check digit algorithms and noted that the Luhn algorithm will detect 7 out 10 twin errors excluding 22 to 55, 33 to 66 and 44 to 77. Also, Luhn formula will not detect the adjacent transposition of digits 09 for 90 or vice

versa. Variations have been made of the Luhn algorithm such as one for alphanumeric f elds. Ghandhi concludes that there is need for an improved error detection mechanism that addresses these weaknesses since human error is unavoidable.

Causley (2012) investigated the Luhn formula and found that if a single digit a is picked such that $a \in \mathbb{Z}_{10}$ and repeat it until the Luhn-check passes, 8888 is the smallest and 6666666666666 is the largest. Also, every single digit error is found regardless of the value or position. Checksum *mod* 10 also have this attribute but Luhn formula has an extra perk in that almost all transposition of adjacent digits are found. The only undetected case is when switching the position of 0 and 9, which computes to 2% of all cases.

Mohr (2015) analyzed the various methods that have been used in check digit calculations including their strength and weaknesses. Among this methods is the Luhn algorithm or the IBM Check which is described as the even or odd weighted algorithm. The algorithm detects all the single digit errors and adjacent transpositions but not jump transposition (such as 553 becoming 355) or 09 becoming 90. Mohr has not given the reason as to why the Luhn formula fails in detection of these errors.

Raymond (1986) states that in a f eld there are no zero divisors, that is the product of any two non-zero elements is a non zero element, which holds in ISBN modulo 11, but does not hold for modulo 10 since $\exists\ a \in \mathbb{Z}_{10}$ such that $(a, 10) \neq 1$. For any digit $a \in \mathbb{Z}_{10}$ such that $a = 2n, n \in \mathbb{Z}$ when multiplied by 5 it gives a product which is a multiple of 10 making them to be zero divisors of $\mathbb{Z}_{10}$.

Gallian (1991) states that there exists no modulus 10 scheme that detects all single digits and all transposition errors. If a modulus 10 scheme detects all single digit errors then there exist transposition errors which remain undetected by the scheme. Also, Gallian gave the condition necessary for the detection of most errors that commonly occur during data transmision. These conditions will be used to analyse the Luhn formula so as to f nd the weaknesses in the Luhn algorithm.

Hussein (2013) observed that many card numbers exist and at such volumes the Luhn algorithm cannot distinguish among these numbers. A lot of test shows that the Luhn algorithm suffers from weaknesses including the failure to determine the length and type of credit card number being analyzed. Luhn algorithm checks only the check digit which happens to be the last digit of a credit card number while neglecting length of the number under discussion. For example a master card number (5578249275041923) is valid according to the Luhn algorithm and on deleting the last three digits from the rightmost (5578249275041) the number still passes as valid despite the fact that it is invalid. Hussein *et al*. proposed an enhanced Luhn algorithm f ow chart that will determine the length and type of credit card number.

Vasiga (2008) stated that the credit card is an $n-$digit sequence $d_1, d_2, d_3, \ldots, d_n$ where

each $d_i \in \mathbb{Z}_{10}$ and satisfies the check digit equation: $d_n = 10 - \left(\sum_{i=1}^{n-1} d_i \right) \ mod\, 10$. That ensures the check digit is dependent on the $n-1$ digits. Also, the transmission of credit card digits from point A to point B can be altered with the probability ε where each digit d_j $(1 \leq j \leq n)$ is changed (with uniform distribution) to a different digit chosen from the set $\{0,1,2,\ldots,9\} - \{d_j\}$.

Causley (2012) discovered subtracting 9 when the checksum digit is greater than 10 is much simpler than when adding the digits to get a single digit. This will not affect the result when the checksum is divided by 10. He also calculated the total number of distinct cards before the credit card company runs out of numbers. A credit card number has 16 digits. The first 15 digits can be arranged in any order but the 16^{th} digit which is the check digit is determined by the other numbers. He concludes there are 10^{15} such arrangements.

Kamaku (2012) analyzed how the total number of code words in $ISBN-10$ and the $ISBN-13$ affects the error detection and correction capabilities by showing that both codes have major weaknesses in regard to the properties mentioned. Kamaku proposed a code, the $ISBN-16$, which has a larger dictionary and a better error detection and correction capability than the $ISBN-13$.

Gallian (1991) provides another method in which someone can use to validate credit cards through the use of permutations $\sigma = (0)(124875)(36)(9)$. For any string of digits $a_1a_2...a_{n-1}$ we assign a check digit a_n so that $\sigma(a_1)+a_2+\sigma(a_3)+a_4+\ldots+\sigma(a_{n-1})+a_n \equiv 0\,(mod\ 10)$. The use of permutations does not explain the reason why some errors can not be detected by the Luhn formula.

Tervor (2014) showed that Luhn formula is not perfect in detection of swapped digits. Although swapping two adjacent digits will generally result in a checksum failure, there are cases in which it will not be detected. If the digits 0 and 9 are side-by-side and they get swapped their contribution to the total sum will be the same. That is:

$$\begin{array}{cccc} \cdots & 0 & 9 & \cdots \\ \cdots & \times 1 & \times 2 & \cdots \\ \cdots & 0 & 18 & \cdots \\ \cdots & 0 & 9 & \cdots \end{array} \quad \text{and} \quad \begin{array}{cccc} \cdots & 9 & 0 & \cdots \\ \cdots & \times 1 & \times 2 & \cdots \\ \cdots & 9 & 0 & \cdots \\ \cdots & 9 & 0 & \cdots \end{array}$$

Thus, the contribution of digits 0 and 9 does not change with position. Trevor claimed that for all possible two digit values (00..99) there are 90 cases of two different digits side by side and that the algorithm will fail to detect swapped digits in two of these cases ($'09'$ becomes $'90'$, and $'90'$ becomes $'09'$). Consequently, swapped digits will be detected 88 times out of 90 (97.8%).

Tervor (2014) also discovered that if two non-adjacent digits are swapped, either the change is not detected at all (if they are both multiplied by 1, or both by 2), or the situation is identical to swapping adjacent digits. Tervor concludes that a systematic approach to error detection could lead to better codes.
Rane (2015) described an algorithm better than the Luhn formula called the Verhoeff Scheme which uses permutation and non-commutative dihedral group operations (D_5) to evaluate the check digit. Even though the algorithm detects all single errors and all adjacent transposition errors it does not however detect all types of jump transposition. Despite Verhoeff scheme being better in error detection than the Luhn formula, it has not been widely used since it has complicated computation.
Therefore, there is need to analyse the Luhn algorithm in order to determine it's weaknesses in as far as error detection is concerned.

1.3 Statement of the Problem

Luhn algorithm will detect all single digit errors and almost all transposition of adjacent digits. It does not however detect the transposition of digits 09 and 90, twin error $22 \leftrightarrow 55$, $33 \leftrightarrow 66$ and $44 \leftrightarrow 77$ and jump transposition of the form $abb \leftrightarrow bba$. Also, Luhn formula has no error correction capabilities. In this research we give the weaknesses in the Luhn algorithm that makes it not effective in error detection and give the conditions necessarry for detection of some of those errors. Also an enhanced algorithm is proposed that addresses those weaknesses.

1.4 Justif cation

The world's population is rapidly growing, hence there is need for a proper mechanism for eff cient data storage and retrieval of information without expensive human intervention. Luhn formula is widely used in this area of data storage and retrieval, mostly for security and business purposes. Credit cards (visa cards, master cards, discovery cards) commonly referred as "plastic money", is one of those gadgets that apply the Luhn formula in its validation before it is used for transaction. Fraudsters and conmen have tried to crack the formula to their own advantage, and by them discovering that the formula has weaknesses, our cards will no longer be safe. In the long run this will make customers using credit cards lack conf dence in its eff ciency. Therefore, there is a need to create an improved error detection and correction formula.
Other applications of the Luhn formula is in the *IMEI* (International Mobile Equipment Identity) number used for unique identif cation of mobile phones, National Provider

Identif er numbers in US and Canadian Social Insurance Numbers.
The Luhn formula is still widely used in many industries as described above probably because it is simple and is avaible for the public domain. The following chapters will elaborate on the weaknesses of the formula which will necessitate the need for a more improved algorithm.

1.5 Objectives

1.5.1 General Objective

This research seeks to determine and analyze the error detection capabilities of the Luhn formula on the credit card number and design a new algorithm that addresses the weaknesses.

1.5.2 Specif c Objectives

1. To determine the effectiveness and weaknesses of the use of *modulo* 10 in the Luhn algorithm.
2. To determine the weaknesses and conditions necessary for the detection of transposition of digits 09 for 90.
3. To determine the weaknesses and conditions necessary for the detection of twin errors $22 \leftrightarrow 55$, $33 \leftrightarrow 66$ and $44 \leftrightarrow 77$.
4. To determine the weaknesses and conditions necessary for the detection of jump transposition.
5. To develop an algorithm that addresses the weaknesses of the Luhn algorithm.

CHAPTER TWO
MODULUS ON LUHN FORMULA

2.1 Introduction

This chapter analyzes the relationship between the Luhn formula and its modulus. According to Gallian (1991), a prime modulus is more effective in error detection than a composite modulus. The properties of a f nite f eld will be used to determine the effectiveness of the modulus 10 in error detection in regard to the Luhn formula. Theorem 1.1 will be used to show that modulo 10 is not a f eld and therefore not effective in error detection.

2.2 Modulus 10

Theorem 2.1. *Let $i \in \mathbb{N}$, and p_i the product of code word digits with their respective weights and $m, q \in \mathbb{Z}_{10}$ where $m_i = p_i \, mod \, 9$. Then the checksum $= \sum_{i=1}^{n} p_i + \sum_{i=1}^{n} m_i + \sum_{i=1}^{n} 9q_i$ in Luhn algorithm*

Proof. After multiplying the code word digits with the weights, if the product $p \geq 10$, the digits of the product are added together to produce a single digit, that is $\forall \, p \geq 10 | \, p = ab$ add $a + b = c$ where $\forall \, a, b, c \in \mathbb{Z}_{10}$, is the same as calculating $p_i \, mod \, 9$ where $9 \nmid p_i$ or $\forall \, p_i > 9$ and $9 | p_i$ calculate $p_i - 9$ or for $p_i \leq 9$ and $9 | p_i$ then p_i.
Therefore, if $9 \nmid p_i$ then $p_i \, mod \, 9 = m_i$, $\forall \, m_i \in \mathbb{Z}_9$ and if $9 | p_i$and $p_i > 9$ then $p_i - 9 = 9q_i$, $\forall \, q_i \in \mathbb{Z}_{10}$. Otherwise for all $p_i \leq 9$ and $9 | p_i$ then p_i. This makes the checksum of a code word using Luhn algorithm to be $\sum_{i=1}^{n} p_i + \sum_{i=1}^{n} m_i + \sum_{i=1}^{n} 9q_i$ □

From Theorem 2.1 one can easily validate a code word using Luhn algorithm by applying $\left(checksum = \sum_{i=1}^{n} p_i + \sum_{i=1}^{n} m_i + \sum_{i=1}^{n} 9q_i \right) \, mod \, 10$. If the solution is 0 then the code word is valid. In calculating the check digit one can apply
$10 - \left(checksum = \sum_{i=1}^{n} p_i + \sum_{i=1}^{n-1} m_i + \sum_{i=1}^{n-1} 9q_i \right) mod \, 10$
where the solution is in the set $\{0, 1, 2, ..., 9\} = \mathbb{Z}_{10}$. If there is no digit in the code word such that $9 | p_i$ then one can claim that the checksum is $\sum_{i=1}^{n} m_i$ consequently for validation of a code word $\left(checksum = \sum_{i=1}^{n} m_i \right) \, mod \, 10$ and $10 - \left(checksum = \sum_{i=1}^{n} m_i \right) \, mod \, 10$ for check digit calculation. Thus if $9 \nmid p_i$ then Luhn algorithm uses modulo 9 and modulo 10 in error detection.

Theorem 2.2. *The usage of more than one modulus in Luhn algorithm does not guarantee all error detection.*

Proof. By counter example.
From Theorem 2.1 it is shown that Luhn formula uses two moduluses for error detection if $9 \nmid p_i$, where p_i is the product of the code word digits with its respective weight, that is modulo 9 and modulo 10. Twin error $22 \rightarrow 55$ will be considered

Table 2.1: Twin Error Validation

number a	2	2
aw_i	2	4
$(aw_i)\,mod\,9$	2	4

and

number b	5	5
bw_i	5	10
$(bw_i)\,mod\,9$	5	1

Note that $\left[\left(\sum_{i=1}^{n} aw_i\right) mod\,9\right] mod\,10 = \left[\left(\sum_{i=1}^{n} bw_i\right) mod\,9\right] mod\,10$. Hence usage of more than one modulus does not guarantee detection of all errors in Luhn algorithm. □

Theorem 2.3. *The set generated by the Luhn formula does not form a f nite f eld.*

Proof. Suppose it forms a f nite f eld with elements $\{0,1,...,9\}$.
Then for all $a \in \mathbb{Z}_{10}$, $gcd\ (a,10) = 1$, hence there are no zero divisors of $\mathbb{Z}_{10}$.
This implies that $\nexists b \in \mathbb{Z}_{10}$ such that for all even number $d \in \mathbb{Z}_{10}$ then $10|\,(b.d)$, which is false, since there exist zero divisors in modulo 10 which yield multiples of 10 on multiplication. Therefore, the set generated by the Luhn formula does not form a f nite f eld.
Alternatively, suppose $\mathbb{Z}_{10}$ is a f nite f eld.
Let $m = a \cdot b = 10k$ where $k \in \mathbb{Z}$ and $1 < a,b < 10$. $\forall\, a \in \mathbb{Z}_{10}\, \exists\, a^{-1} \in \mathbb{Z}_{10}$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$.
$b = 1 \cdot b = (a^{-1} \cdot a)b = a^{-1} \cdot (a \cdot b) = a^{-1} \cdot m = 0\, mod\ 10$.
Hence the set generated by the Luhn formula does not form a f nite f eld. □

Proposition 2.1. *The Luhn formula detects all single digit errors.*

Proof. Let a_i be a digit at the i^{th} position which is mistaken for digit a_j and w_i be the weight at the i^{th} position. This theorem will be proved using two cases;
when $w_i = 1$ and when $w_i = 2$.

Case 1. When $w_i = 1$.
In this case aw_i will always be less than 10, that is $aw_i = a$ where $a \in \mathbb{Z}_{10}$.

$\forall a \in \mathbb{Z}_{10},\ a_i - a_j \leq 9 \Longrightarrow \left[(a_i - a_j)\, w_i\right] \ mod\, 10 \neq 0.$
The solution set for this case is $\{0,1,2,...,9\}$ of which all of this digits are less than 10. Therefore Luhn formula detects all single digit errors whenever $(w_i, 10) = 1$ or when $w_i = 1$.

Case 2. When $w_i = 2$.
In this case, if $aw_i \geq 10$ calculate $aw_i \, mod\ 9$ if $9 \nmid p_i$ or calculate $aw_i - 9$ if $9|aw_i$ and $aw_i > 9$ otherwise aw_i as shown in Theorem 2.1 . Note that $(w_i, 10) \neq 1$.
If both a_i and a_j are less than 4, $a_i - a_j \leq 4 \Longrightarrow \left[(a_i - a_j)\, w_i\right] \ mod\, 10 \neq 0$, since all $aw < 9$.
If $0 \leq a_i \leq 4$ and $5 \leq a_j \leq 9$, $a_i - a_j < 0$. $\forall a \in \mathbb{Z}_{10}$ since if $9 \nmid aw_j$ calculate $aw_j \, mod\ 9$ and if $aw_j \geq 10$ and $9|aw_j$then calculate $aw_j - 9$. Therefore, $\left[a_i w_i - a_j w_i\right] \ mod\, 10 \neq 0$.
If $5 \leq a_i \leq 9$ and $0 \leq a_j \leq 4$, $a_i - a_j \leq 9$. $\forall a \in \mathbb{Z}_{10}$ if $aw_i \neq 9k, k \in \mathbb{Z}$ calculate $aw_i \, mod\ 9$ and if $aw_i \geq 10$ and $aw_i = 9k$ calculate $aw_i - 9$.Therefore, $\left[a_i w_i - a_j w_i\right] \ mod\, 10 \neq 0$.
If $5 \leq a_i, a_j \leq 9$, $a_i - a_j \leq 4 \Longrightarrow \left[a_i w_i - a_j w_i\right] \ mod\, 10 \neq 0$. Thus all single digit errors are detectable except when $a_i = a_j$.

□

From Proposition 2.1 it is seen that Luhn algorithm detects all single digit errors which is highly contributed to the usage of modulus 9 and modulus 10 as seen in Theorem 2.1

Theorem 2.4. *In Luhn formula* $\left[(a_i - a_j) w_i\right] \ mod\, 10 = 0$ *iff* $a_i = a_j \ \forall a \in \mathbb{Z}_{10}$.

Proof. If $\left[\left(a_i - a_j\right) w_i\right] \ mod\, 10 = 0$ then $a_i = a_j$ has been proved by proposition 2.1
Conversely, suppose $a_i = a_j$
$a_i - a_j = 0$ implying that $a_i - a_j = 0 w_i = 0 \, (mod\, 10)$.
Hence Luhn algorithm is effective in detection of all single digit errors errors. □

Theorem 2.5. *If an error detecting scheme is effective in error detection then the modulus is prime.*

Proof. By contraposition, suppose the modulus is composite, that is, if n is the modulus then $n = a \cdot b$, where $1 < a, b < n$.
Let the weight factor at the i^{th} position be $\alpha_i = a \cdot k$, $k \in \mathbb{Z}$ and an error occurs such that the digit d, $d \in \mathbb{Z}_n$ at the i^{th} position is mistaken for digit b.
Therefore, $\alpha_i \cdot b = (a \cdot k) \cdot b = k \cdot (a \cdot b) = k \cdot n \equiv 0 \ mod\ n$, hence does not detect all errors and thus not effective in error detection.

Using a prime modulus, zero divisors do not exist, thus the errors are detected. Thus an error detecting scheme with a prime modulus is more effective in error detection. □

Therefore, modulo 10 and modulo 9 being composite modulus used by the Luhn formula are not effective in error detection since even though it detects all single digit errors it however allows some twin errors and transposition errors to pass undetected.

CHAPTER THREE
WEIGHTS VERSUS THE MODULUS

3.1 Introduction

This chapter analyzes the Luhn formula in detection of the transposition of digits 09 for 90 , twin error $22 \longleftrightarrow 55$, $33 \longleftrightarrow 66$, $44 \longleftrightarrow 77$ and jump transposition of the form $abb \longleftrightarrow bba$. It is shown how the relation between the modulus and the weights used affect the ability of the algorithm in detection of the various types of errors.

3.2 Relationship between weights and the modulus

As discussed in Chapter 1 the Luhn algorithm uses modulo 10 and weights of 2 and 1 in error detection where most errors are captured while others remain undetectable. According to Gallian (1991), the relationship between the weights and modulus used in a formula determines a lot in the effectiveness of an error detecting scheme and its optimality in error detection. The following analyzes why the Luhn formula fails in detection of some of its errors and if it complies with the rules as stated by Gallian of how to obtain the optimum checksum scheme.

3.3 Transposition error detection

3.3.1 Transposition error of digits 0 and 9

In this subsection all theorems and transpositions are in reference to the transposition of digits 09 for 90.

Theorem 3.1. *Let $a_i, a_j \in \mathbb{Z}_{10}$ be two digits in the i^{th} and j^{th} position that are transposed in a code word and w_i and w_j their respective weights, then the product $(a_i - a_j)(w_i - w_j)$ is not always distributive i.e. $(a_i - a_j)(w_i - w_j) \neq a_i w_i - a_j w_i - a_i w_j + a_j w_j$ in Luhn algorithm.*

Proof. By counter example : if $a_i = 9, a_j = 0, w_i = 2, w_j = 1$.
$(9-0)(2-1) = 9 \times 1 = 9\,(mod\ 10) \neq 0$.....1
$9 \cdot 2 - 0 \cdot 2 - 9 \cdot 1 + 0 \cdot 1 = 18 - 9 = 9 - 9 = 0\,(mod\,10) = 0$ 2
Hence 1≠2.
Since Equation 1 ≠ Equation 2 the trasposition of digits 09 for 90 are undetected by the Luhn algorithm and vice versa. Reducing the product to a single digit when $aw \geq 10$ makes eqn 2 to be zero, thus not equal to eqn 1. Making the equation
$(a_i - a_j)(w_i - w_j) \neq a_i w_i - a_j w_i - a_i w_j + a_j w_j$ in Luhn algorithm, *i.e.* not distributive.
□

From Theorem 2.1, reduction of the digits of the product $p > 9$ to a single digit makes the distributive law not to hold for all types of transposition errors. Luhn formula will always detect all types of transposition errors as long as the distributive law is maintained.

Theorem 3.2. *Let w_i and w_j, where $w_i \neq w_j$ be weights in a code word working modulo 10. The gcd $(w_i - w_j, 10) = 1$ doesn't guarantee detection of all adjacent transposition errors in the Luhn formula.*

Proof. By counter example, suppose $a_i = 9$, $a_j = 0$, $w_i = 2$, $w_j = 1$. Note that $w_i - w_j = 1 \Longrightarrow \gcd(w_i - w_j, 10) = 1$. From Theorem 3.1 it is shown that $a_i w_i - a_j w_i - a_i w_j + a_j w_j = 0 \ (mod 10)$. (Since when $\forall\ aw \geq 10$ calculate $aw\ mod\ 9$ if $9 \nmid aw$ or calculate $aw - 9$ if $9|aw$ otherwise aw), which makes transposition error 09 for 90 not to be detected by the formula. Hence gcd $(w_i - w_j, 10) = 1$ does not guarantee detection of all adjacent transposition error in the Luhn formula. □

Theorem 3.3. *Let $a, b \in \mathbb{Z}_{10}$ be digits at the i^{th} and j^{th}position and w_i and w_j, $w_i \neq w_j$, their respective weights, if Luhn formula detects all transposition error of the form ab for ba then $\forall\, a, b \in \mathbb{Z}_{10}$, $9 \nmid (aw_i - bw_j)$.*

Proof. By contraposition, suppose $9 \mid (aw_i - bw_j)$. That means $aw_i - bw_j = 9k$, $k \in \mathbb{Z}$. If $w_i = 2$ and $w_j = 1$, there are only two instances when $aw_i - bw_j = 9k$. That is when:

Case 1. When $a = 0$ and $b = 9$. $aw_i - bw_j = 0 - 9 = -9 = 9(-1)$, hence a multiple of 9. From Theorem 3.1 Luhn formula does not detect transposition of digit 09 for 90. Thus Luhn formula does not detect all transposition of digit *ab* for *ba*. This implies the contraposition is true, that is, Luhn formula detects transposition of digit $ab \rightarrow ba$ if $9 \nmid (aw_i - bw_j)$.

Case 2. When $a = 9$ and $b = 0$. $aw_i - bw_j = 18 - 0 = 9 - 0 = 9 = 9(1)$, since $\forall\, aw \geq 10$ calculate $aw\ mod\ 9$ if $9 \nmid aw$ or calculate $aw - 9$ if $9|aw$ to obtain a single digit. Hence $aw_i - bw_j$ is a multiple of 9. From Theorem 3.1 Luhn formula does not detect transposition of digit 90 for 09. Thus, Luhn formula does not detect all transposition of digit *ab* for *ba*. This implies the contraposition is true, that is, Luhn formula detects transposition of digit $ab \rightarrow ba$ if $9 \nmid (aw_i - bw_j)$.

□

Theorem 3.4. *Let $a \in \mathbb{Z}_{10}$, $1 \leq a \leq 4$ and $w_i = 2$, $w_j = 1$ be weights at the i^{th} and j^{th} position respectively. Suppose an adjacent transposition of the form $a0 \longrightarrow 0a$ occurs at the i^{th} and j^{th} position then $aw_i - aw_j = a$ in Luhn formula.*

Proof. $\forall a \in \mathbb{Z}_{10}$, $1 \leq a \leq 4$, $aw_i = 2a < 9$ while $aw_j = a \leq 4$, implying that $aw_i > aw_j$ since $2a > a$, hence $aw_i + 0w_j - 0w_i - aw_j = aw_i - aw_j = 2a - a = a$.
Thus for $1 \leq a \leq 4$, $aw_i - aw_j = a$ in Luhn formula. □

Theorem 3.5. *For all $a \geq 5$, $a \in \mathbb{Z}_{10}$ where $w_i = 2$ and $w_j = 1$, $aw_i - aw_j \leq 0$ in Luhn algorithm.*

Proof. From Theorem 2.1 reducing aw_i to a single digit makes it to be in the range $0 \leq aw_i \leq 9$. Since $w_j = 1$, this implies that aw_i will be in the range $5 \leq aw_j \leq 9$ implying that $aw_j \geq aw_i$. Thus, $aw_i - aw_j \leq 0$. □

Luhn formula detects all transposition errors of the form $ab \longrightarrow ba$ except when $9 | \left(aw_i - aw_j\right)$. There is only one instance when $9 | \left(aw_i - aw_j\right)$, that is, when $a = 9$. From Theorem 3.2, it is clearly shown that Luhn formula detects all types of transposition errors except when 0 and 9 are transposed.

Corollary 3.1. *Let $a \in \mathbb{Z}_{10}$ be a digit in a codeword and $w_i = 2$, $w_j = 1$ be weights at the i^{th} and j^{th} position respectively. Luhn formula does not detect transposition errors of the form $a0 \longrightarrow 0a$ if $9 \mid \left(aw_i - bw_j\right)$.*

Proof. From Theorem 3.4, it is shown that $\forall a \in \mathbb{Z}_{10}$, $1 \leq a \leq 4$, $aw_i - aw_j = a$ and $9 \nmid a$ since a is always less than 4. Also, for $a \geq 5$ and $aw_i - aw_j < 0$, Luhn formula detects all types of adjacent transposition except when $aw_i - aw_j = 0$ which only happens when $a = 9$. Since $9 \mid 0$ from Theorem 3.2, Luhn formula does not detect transposition of digits 0 and 9. Thus, Luhn formula does not detect adjacent transposition of the form $a0 \longrightarrow 0a$ if $9 \mid \left(aw_i - bw_j\right)$. □

In this subsection, it can be seen clearly that addition of products digits $p > 9$ has highly contributed to the reason why the Luhn algorithm does not detect adjacent transposition of digit 0 and 9. By reducing the product $aw_i = p = 18$ where $a = 9$ and $w_i = 2$ to a single digit one gets digit 9 which is equal to product $aw_j = p = 9$ where $a = 9$ and $w_j = 1$. This makes $(a - b)\left(w_i - w_j\right)$ not to be distributive in addition to the reason why $9 \mid \left(aw_i - bw_j\right)$ consequently the reason why 0 and 9 are the only adjacent digits which when transposed cannot be detected by the Luhn algorithm.

3.3.2 Jump transposition

Theorem 3.6. *Let $a_i, a_j \in \mathbb{Z}_{10}$ where $a_i \neq a_j$, $i, j \in \mathbb{N}$ be digits in a code word at positions $2i-1$ and $2i+1$ respectively which undergoes jump transposition of the form $a_i a_j a_j \longrightarrow a_j a_j a_i$. Let w_i and w_j be the weights at position $2i-1$ and $2i+1$ respectively. Then $(a_i - a_j)(w_i - w_j) = 0 \,(mod\, 10)$ if and only if $w_i = w_j$.*

Proof. Consider jump transposition of the form $a_i a_j a_j$ for $a_j a_j a_i$ in the Luhn algorithm.

Table 3.1: Jump Transposition

a_i	a_j	a_j	$\longrightarrow$	a_j	a_j	a_i
w_i	w_k	w_i		w_i	w_k	w_i

From Table 3.3.2, observe that the transposed digits a_i, $a_j \in \mathbb{Z}_{10}$ have the same weights at position $2i-1$ and $2i+1$ in Luhn formula implying that $w_i = w_j \Longrightarrow w_i - w_j = w_i - w_i = 0$. This implies that $0 = (a_i - a_j)0 = (a_i - a_j)(w_i - w_i) = a_i w_i - a_j w_i - a_i w_i + a_j w_i = 0 = 0\,(mod\,10)$. □

Conversely, suppose $(a_i - a_j)(w_i - w_j) = 0\ (mod\,10)$. Since $a_i \neq a_j \Longrightarrow a_i - a_j \neq 0$ hence $w_i - w_j = 0$. Consequently, $w_i = w_j$. Therefore Luhn formula does not detect jump transposition of the form *abb* for *bba* if the weights at the $2i-1$ and $2i+1$ or the $2i$ and $2i+2$ position are equal

Corollary 3.2. *Let $a,b,c \in \mathbb{Z}_{10}$, where $a \neq b \neq c$ be digits at positions $2i-1$, $2i$ and $2i+1$ respectively and suppose $\alpha_i = 2$ is the weight at position $2i-1$. The Luhn formula does not detect jump transposition of the form $abc \longrightarrow cba$ if and only if the weights at positions $2i-1$ and $2i+1$ are equal.*

Proof. The weight position $2i-1$ is the same as the weight at position $2i+1$ in Luhn algorithm as seen in Theorem 3.6, that is they are all equal to $\alpha_i = 2$ implying that the weight at position $2i$ is equal to one as seen in Theorem 3.6,

$$0 = (a-c)\cdot 0 = (a-c)\cdot(\alpha_i - \alpha_i) = a\cdot\alpha_i - c\cdot\alpha_i - a\cdot\alpha_i + c\cdot\alpha_i = 0 \equiv 0\,(mod\,10).$$

Thus, Luhn formula does not detect jump transposition of the form $abc \longrightarrow cba$ if the weights at positions $2i-1$ and $2i+1$ are equal.
Conversely, suppose Luhn formula detects Jump transposition of the form $abc \longrightarrow cba$ where $a \neq b \neq c$, then the weights at position $2i-1$ and $2i+1$ are not equal. That is , if α_i and α_k are the weights at position $2i-1$ and $2i+1$ respectively, then

$$[a\alpha_i + c\alpha_k - (c\alpha_i + a\alpha_k)]\ mod\,10 \neq 0 \Longrightarrow [a\alpha_i + c\alpha_k - c\alpha_i - a\alpha_k]\ mod\,10$$

$$= [(a-c)\,\alpha_i - \alpha_k\,(a-c)]\ mod\,10 \neq 0$$

This implies that $[(a-c)(\alpha_i - \alpha_k)]\ mod\,10 \neq 0$.
Since $(a-c) \neq 0$, also $(\alpha_i - \alpha_k) \neq 0$ implying that $\alpha_i \neq \alpha_k$.

Hence the contraposition that Luhn formula does not detect jump transposition of the form $abc \longrightarrow cba$ if $\alpha_i = \alpha_k$ is true since the weights at position $2i-1$ and $2i+1$are equal. □

Theorem 3.7. *Let $a_1a_2...a_n$, where $n = 2k+1$, $k \in \mathbb{Z}$ be digits of a codeword.*

1. Luhn formula does not detect transpostion of even positioned digits.

2. Luhn formula does not detect transpostion of odd positioned digits.

Proof. Let a_i and a_j be digits at the i^{th} and j^{th} position (even positions) where $a_i \neq a_j$ that are transposed in a code. The weights α_i and α_j are equal since they are both evenly positioned. Hence they are all equal to 2. This implies that
$a_i\alpha_i + a_j\alpha_j \equiv a_j\alpha_i + a_i\alpha_j (mod\ 10) \Longrightarrow a_i\alpha_i + a_j\alpha_j - (a_i\alpha_j + a_j\alpha_j) = 0 \equiv 0\ (mod\ 10)$
since $\alpha_i = \alpha_j$.
Thus Luhn formula does not detect transposition of even positioned digits in a code word if $n = 2k+1$.
Also, let a_i and a_j be digits at the i^{th} and j^{th} position (odd positions) that are transposed in a code. The weights α_i and α_j are equal since they are both oddly positioned. Hence they are all equal to 1. This implies that

$$a_i\alpha_i + a_j\alpha_j \equiv a_j\alpha_i + a_i\alpha_j\,(mod\,10) \Longrightarrow a_i + a_j \equiv a_i + a_j\,(mod\,10)$$

$$\Longrightarrow a_i + a_j - (a_i + a_j) \equiv 0\ (mod\,10)$$

Thus Luhn formula does not detect transposition of odd positioned digits in a code word if $n = 2k+1$. □

Corollary 3.3. *Let $a_1a_2...a_n$, where $n = 2k$, $k \in \mathbb{Z}$ be digits of a codeword.*

1. Luhn formula does not detect transpostion of even positioned digits.

2. Luhn formula does not detect transpostion of odd positioned digits.

Proof. The proof will follow from Theorem 3.7 □

Theorem 3.8. *Let $a, b, c \in \mathbb{Z}_n$, $n \in \mathbb{N}$, where $a \neq b \neq c$ be digits in a code word at positions i, j, and k respectively and w_i, w_j and w_k, where $w_i = w_k \neq w_j$ be the weights at the respective positions. If an error detecting scheme detect jump transposition of the form $abc \longrightarrow cba$ then $\left[(abc)\cdot(w_iw_jw_k) - (cba)\cdot(w_iw_jw_k)\right]$ mod $n \neq 0$.*

Proof. By Contraposition, suppose $\left[(abc)\centerdot\left(w_iw_jw_k\right)-(cba)\centerdot\left(w_iw_jw_k\right)\right]\equiv 0\ (mod\ n)$,

$$n|\left[(abc)\centerdot\left(w_iw_jw_k\right)-(cba)\centerdot\left(w_iw_jw_k\right)=0\right]$$

$$\Longrightarrow (abc)\centerdot\left(w_iw_jw_k\right)=(cba)\centerdot\left(w_iw_jw_k\right)$$

This can only happen if $w_i=w_k$ since $a\neq c$. Hence an error detecting scheme can not detect jump transposition of the form $abc\longrightarrow cba$ if $(abc)\centerdot\left(w_iw_jw_k\right)-(cba)\centerdot\left(w_iw_jw_k\right)\equiv 0 mod n$. Thus the contraposition is true. □

In this subsection, it is clear that Luhn formula cannot and will never detect any form of Jump transposition since the weights at the position $2i-1$ and $2i+1$ or at position $2i$ and $2i+2$ will always be equal. In order to detect this type of transposition error then the weights at the affected positions should not be equal or otherwise they will contribute the same checksum and consequently the same check digit when multiplied with code word digits.

3.3.3 Twin errors.

According to Ghandhi (2015), Luhn formula does not detect 3 out of 10 twin errors. These twin errors are $22\longleftrightarrow 55$, $33\longleftrightarrow 66$, and $44\longleftrightarrow 77$. This section will determine the weaknesses in the Luhn formula that reduces its effectiveness in twin error detection and the conditions necessary for it to detect any type of twin error.

Theorem 3.9. *Let $a, b\in\mathbb{Z}_{10}$, $1\leq a\leq 4$ and $5\leq b\leq 9$ and w_i be the weight used. Luhn formula detects all twin errors of the form $aa\longrightarrow bb$ if*

$$\begin{cases} aw_i-bw_i\neq -3 & when \quad w_i=1 \\ aw_i-bw_i\neq 3 & \qquad\quad w_i=2 \end{cases}.$$

Proof. By Contraposition, suppose Luhn formula does not detect all twin errors of the form $aa\longrightarrow bb$. This implies that both twins have the same checksum contribution, that is,

$$aw_i+aw_j=bw_i+bw_j$$

$$aw_i-bw_i=bw_j-aw_j$$

The table 3.2 is a table of aw_i-bw_i when $w_i=1$. □

Table 3.2: Twin Error Detection when $w_i = 1$

aw_i \ bw_i	5	6	7	8	9
1	−4	−5	−6	−7	−8
2	−3	−4	−5	−6	−7
3	−2	−3	−4	−5	−6
4	−1	−2	−3	−4	−5

$\longrightarrow$

aw_i \ bw_i	5	6	7	8	9
1	√	√	√	√	√
2	×	√	√	√	√
3	√	×	√	√	√
4	√	√	×	√	√

√ *detectable*
× *undetectable*

From Table 3.3.3 above, it is seen that the Luhn formula does not detect twin errors if $aw_i - bw_i = -3$. That is when $a = 2$ and $b = 5$ or when $a = 3$ and $b = 6$ or $a = 4$ and $b = 7$. These are the only twin error undetectable cases in Luhn algorithm. Hence the contraposition is true that Luhn formula detects twin errors of the form $aa \longleftrightarrow bb$ if $aw_i - bw_i \neq -3$ when $w_i = 1$. Without loss of generality, Luhn formula detects twin error of the form $aa \longleftrightarrow bb$ if $bw_j - aw_j \neq -3$ where $w_i \neq w_j = 2$.
Similarly, if $w_i = 2$, the following is a table of $aw_i - bw_i$. Recall that, $\forall\, bw_i \geq 10$ compute $bw_i - 9$ if $9|bw_i$ or compute $bw_i \, mod \, 9$ if $9 \nmid bw_i$ to obtain a single digit.

Table 3.3: Twin error Detection when $w_i = 2$

aw_i \ bw_i	1	3	5	7	9
2	1	−1	−3	−5	−7
4	3	1	−1	−3	−5
6	5	3	1	−1	−3
8	7	5	3	1	−1

$\longrightarrow aw_i$

aw_i \ bw_i	1	3	5	7	9
2	√	√	√	√	√
4	×	√	√	√	√
6	√	×	√	√	√
8	√	√	×	√	√

√ *detectable*
× *undetectable*

From Table 3.3.3, it is shown that Luhn formula does not detect twin errors if $aw_i - bw_i = 3$ when $w_i = 2$. That is when $a = 2$ and $b = 5$ or when $a = 3$ and $b = 6$ or $a = 4$ and $b = 7$. Thus the contraposition is true that Luhn formula detect twin errors of the form $aa \longleftrightarrow bb$ if $aw_i - bw_i \neq 3$. Also, without loss of generality, Luhn formula detects twin errors of the form $aa \longleftrightarrow bb$ if $bw_j - aw_j \neq 3$ where $w_j = 1 \neq w_i$.

Corollary 3.4. *Let $a, b \in \mathbb{Z}_{10}$, $5 \leq a \leq 9$ and $1 \leq b \leq 4$ and w_i be the weight used. Luhn formula detects all twin errors of the form $aa \longrightarrow bb$ if*

$$\begin{cases} aw_i - bw_i \neq 3 & when \quad w_i = 1 \\ aw_i - bw_i \neq -3 & \qquad\quad w_i = 2 \end{cases}.$$

Proof. The proof to be followed from Theorem 3.9. □

Proposition 3.1. *Let w_i and w_{i+1} be weights. $(w_i + w_{i+1}, 10) = 1$ does not guarantee all twin error detection in Luhn algorithm.*

Proof. By counter example, let 22 be two adjacent digits in a code which are replaced by 55. In Luhn algorithm $(w_i + w_{i+1}, 10) = 1$ since $gcd(3, 10) = 1$. Let $w_i = 2$ and $w_{i+1} = 1$, but $(2w_i + 2w_{i+1}) - (5w_i + 5w_{i+1}) = 2 \cdot 2 + 2 \cdot 1 - (5 \cdot 2 + 5 \cdot 1) = 4 + 2 - (10 + 5) = 6 - 1 - 5 \equiv 0 \pmod{10}$ since for any product $p \geq 10$, compute $p - 9$ if ${}^{9}|_{p}$ or compute $p \bmod 9$ if ${}^{9}\nmid_{p}$ to obtain a single digit. This applies for digits $33 \longrightarrow 66$ and digits $44 \longrightarrow 77$. Thus Luhn formula does not guarantee detection of all twin errors even though $(w_i + w_{i+1}, n) = 1$. □

Theorem 3.10. *Let $a, b \in \mathbb{Z}_{10}$ such that $1 \leq a \leq 4$ and $5 \leq b \leq 9$ and w_i and w_j be the weights. If $(a-b) \cdot (w_i + w_{i+1}) = -\left(k^2\right)$, $k \in \mathbb{Z}$ then Luhn formula does not detect twin error of the form $aa \longrightarrow bb$.*

Proof. There are only three instances when $(a-b)(w_i + w_{i+1}) = -\left(k^2\right)$. That is when $a = 2$, $b = 5$ or $a = 3$, $b = 6$ or when $a = 4$, $b = 7$. In all these three cases it is shown in Theorem 3.9 that Luhn formula does not detect twin error of the form $aa \longrightarrow bb$. □

Theorem 3.11. *Let $a, b \in \mathbb{Z}_{10}$ and w_i and w_{i+1} be the weights where $w_i \neq w_{i+1}$. In Luhn formula $(aw_i + aw_{i+1}) - (bw_i + bw_{i+1}) \neq (a-b)(w_i + w_{i+1})$.*

Proof. By counter example, let $a = 3$ and $b = 6$.
$(aw_i + aw_{i+1}) - (bw_i + bw_{i+1}) \equiv 0 \pmod{10}$ since for every aw, $bw \geq 10$, subtract 9 to obtain a single digit.
$[(a-b)(w_i + w_{i+1}) \neq 0] \bmod 10 \neq 0$. Thus, $(aw_i + aw_{i+1}) - (bw_i + bw_{i+1})$ is not always equal to $(a-b)(w_i + w_{i+1})$ in Luhn algorithm. That is, Luhn algorithm is not always distributive. □

From this subsection, Luhn algorithm's weaknesses in regard to twin error detection has been analysed in detail and the conditions stated as to what must be obeyed for the twin errors to be detected.

CHAPTER FOUR
MODULO 13 ALGORITHM

4.1 Introduction

In this chapter, an enhanced algorithm, modulo 13 algorithm is designed, analyzed and discussed. This is an improvement of Luhn modulo 10 algorithm with the aim of possibly eliminating the weakness associated with the Luhn formula. As discussed in the previous chapters, despite Luhn algorithm being a widely used formula, it has weaknesses such as its inability to detect any type of jump transposition errors among others. As long as human beings are part of the communication channel, these errors are inevitable. Modulo 13 algorithm is chosen in consideration to the following properties:

- Effective single error detection
- Effective transposition error detection and jump transposition error detection.
- Effective twin error and jump twin error detection.
- Effective phonetic error detection.

In addition to the above properties, modulo 13 algorithm increases the dictionary size of a 16-alphabet credit cards number.

4.2 Modulo 13 algorithm

As the name suggest, modulo 13 algorithm uses modulo 13 in its error detection. This is a set of 13 distinct digits, that is $F_{13} = \mathbb{Z}_{13} = \{0,1,2,...,9,A,B,C\}$ where the letters A, B and C represent the values indicated in Table 4.1

Table 4.1: Letter Representation in Modulo 13 Algorithm

Letter	A	B	C
Value	10	11	12

Letters are used to avoid confusion when dealing with two digit numbers. For example, consider ...112...which represent some digits in a code word. There are three possibilities in which the digits may be interpreted. These are:

- The sequence represents three independent digits *i.e.*1 followed by 1 then by 2.
- The sequence represents two independent digits *i.e.* 11 followed by 2

- Also, the possibility that the first digit is 1 followed by 12.

Therefore, the letters are used to eliminate any ambiguity.
Modulo 13 is a finite field and thus all digits in the set are co-prime to 13 eliminating possibility of errors associated with zero divisors.
Modulo 13 algorithm will be using weights in a form of decreasing geometric progression in the form 2^{n-i} where $i = \{1, 2, 3, ...\}$ represent the position of a digit in a code word starting from the left most digit. The number 2 in 2^{n-i} has been chosen on the basis that it is the least prime number. This will reduce complications in regards to long calculations, especially when the user is using manual method of calculation.
For example, let $a_1 a_2 ... a_n$ be a code word with n distinct digits. A dot product $(a_1, a_2, ..., a_n) \cdot \left(2^{n-1}, 2^{n-2}, ..., 2^0\right)$ is perfomed to obtain the check sum.
Therefore, suppose $v = a_1 a_2 ... a_{16}$ is a 16-digit credit card number that has to be validated before usage, then it must satisfy the following formula where $i = 1, 2, ..., 16$ and $n = 16$:

$$checksum = \sum_{i=1}^{n} a_i 2^{n-i}$$

for checksum calculation. In order to validate a credit card number then the following condition should be satisfied

$$\left(checksum = \sum_{i=1}^{n} a_i 2^{n-i}\right) \equiv 0 \,(mod\, 13),$$

or

$$\left(checksum = \sum_{i=1}^{n} a_i 2^{n-i}\right) mod\, 13 = 0.$$

In calculation of the check digit, the following formula should be applied

$$checkdigit\ x{=}13 - \left(checksum = \sum_{i=1}^{n} a_i 2^{n-i}\right) mod\ \ 13.$$

For example, consider a visa card number 4111111111111111 that is to be validated

Table 4.2: Example of Visa Card Validation using Modulo 13 Algorithm

Account number (a_i)	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Weight (w_i)	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Product ($a_i w_i$)	131072	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

$$checksum = \sum_{i=1}^{n} a_i w_i = \sum_{i=1}^{n} a_i 2^{n-i} = 163839 \equiv 0\,(mod\ 13)$$

Therefore, the number is valid.

Sometimes, there is need to calculate the check digit (the last digit in a codeword) for error detection purposes.

For example, suppose we have a discovery card number 601101601101601x which a check digit will be added to replace the x digit for it to be complete.

$$checkdigit\ x{=}13 - \left(checksum = \sum_{i=1}^{16} a_i 2^{n-i} = 213250 \right) mod\ 13{=}\ 2$$

Hence the full discovery card number is 6011016011016012

Other examples of modulo 13 algorithm code words include:

4222B678A1112256	$AAAABBBBCCCCAC$	5555444433332222
6543210CBA987654	5111111111111116	8888888888888885
111111111111111C	434566778001223C	4444222211114444B

In the case of an existing code word, modulo 13 algorithm does not have any effect on the code words length, all that changes is the check digit as illustrated in subsection 4.3 below.

4.3 Converting from Luhn modulo 10 to modulo 13 algorithm

Converting code words to modulo 13 algorithm from Luhn modulo 10 in credit cards is easy since both of them use checksum and checkdigit calculation formula. The difference arises in the usage of modulo 13 instead of modulo 10 and weights of $2^{n-1}, 2^{n-2}, ..., 2^0$ are used instead of weights $2, 1, 2, ..., 2, 1$. Also, if the product (of the digits of the code word and the weights) $p > 9$, the product digits are added together to obtain a single digit but in modulo 13 algorithm there is no addition. The check digit in modulo 13 algorithm can be any digit in the set $\{0, 1, 2, ..., A, B, C\}$ where letters A, B and C are digits 10, 11, and 12 respectively.

For example, converting the visa card number using Luhn modulo 10 algorithm to one using modulo 13 algorithm by calculating the check digit. This is achieved by keeping

the f rst 15 digits constant then applying modulo 13 formula on the digits for check digit calculation.
Let a visa card number using Luhn Modulo 10 algorithm be 4459713000181962 that will be converted to modulo 13 algorithm code word.

Table 4.3: Example of Check Digit Calculation in Modulo 13 Algorithm

Account number (a_i)	4	4	5	9	7	1	3	0	0	0	1	8	1	9	6	x
Weight (w_i)	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Product ($a_i w_i$)	131072	65536	40960	36864	14336	1024	1536	0	0	0	32	128	8	144	12	x

$$checkdigit\ x{=}13 - \left(checksum = \sum_{i=1}^{16} a_i 2^{n-i} = 291652 \right) mod\ 13{=}\ 3$$

Therefore, the new code word is 4459713000181963 in modulo 13 algorithm.

4.4 Properties of Modulo 13 Algorithm

4.4.1 Single error detection

Theorem 4.1. *Let a_i be the i^{th} digit of a send code word and suppose b_i is received. Modulo 13 algorithm detects all single digit errors if $[(a_i - b_i)\, w_i] mod\ 13 \neq 0$, $\forall a, b \in \mathbb{Z}_{13}$, $a_i \neq b_i$ and $i \in \mathbb{N}$.*

Proof. The weight at position i is 2^{n-i}. Since $a_i \neq b_i$ this implies that $a_i - b_i \neq 0$. Therefore,$(a_i - b_i)\, 2^{n-i} \neq 0$ hence $\left[(a_i - b_i)\, 2^{n-i}\right] mod\ 13 \neq 0$ since $^{13}\nmid_{a_i - b_i}$ and $^{13}\nmid_{2^{n-i}}$. Hence modulo 13 algorithm detects all single digit errrors. A single digit error will remain undetected if and only if $a_i = b_i$ which in that case there is no error. □

For example, consider the code word 5211111111111116 that is to be validated.
But $\left(checksum = \sum_{i=1}^{16} a_i 2^{n-i} = 212996 \right) \equiv 4\, mod\, 13$. Thus, the code word has a single digit error.
It is not possible to detect the position of the error, therefore, there is need to seek for retransmission or the code word be discarded. Any single error at any position will change the checksum ultimately changing the checkdigit.

4.4.2 Transposition error detection

Theorem 4.2. *Let $n \in \mathbb{N}$ be the number of digits in a code word and $i \in \mathbb{N}$ be the position of a digit, the difference between the weights $2^{n-i} - 2^{n-(i+1)} = 2^{n-(i+1)}$ in modulo 13 algorithm.*

Proof. In modulo 13 algorithm, the weights used are of the form
$2^{n-1}, 2^{n-2}, 2^{n-3}, \cdots, 2^{n-i}, 2^{n-(i+1)}, \cdots, 2^1, 2^0$.
$2^{n-i} - 2^{n-(i+1)} = 2^{n-i} - 2^{n-i-1} = 2^{n-i} - 2^{n-i} \cdot 2^{-1} = 2^{n-i}\left(2^0 - 2^{-1}\right) = 2^{n-i} \cdot 2^{-1} = 2^{n-i-1} = 2^{n-(i+1)}$.
Thus $2^{n-i} - 2^{n-(i+1)} = 2^{n-(i+1)}$ in modulo 13 algorithm. This will be used in Theorem 4.4 to show that modulo 13 algorithm detects all types of adjacent transpostion errors. □

Theorem 4.3. *Let $n \in \mathbb{N}$ be the number of digits in a code word and $i \in \mathbb{N}$ be the position of a digit, the difference between the weights $2^{n-i} - 2^{n-(i+2)} = 3 \cdot 2^{n-(i+2)}$ in modulo* 13 *algorithm.*

Proof. In modulo 13 algorithm, the weights used are of the form
$2^{n-1}, 2^{n-2}, 2^{n-3}, \cdots, 2^{n-i}, 2^{n-(i+1)}, 2^{n-(i+2)}, \cdots, 2^1, 2^0$.
$2^{n-i} - 2^{n-(i+2)} = 2^{n-i} - 2^{n-i-2} = 2^{n-i} - 2^{n-i} \cdot 2^{-2} = 2^{n-i}\left(2^0 - 2^{-2}\right) = 2^{n-i} \cdot (1 - 1/4) = 2^{n-i} \cdot (3/4) = 3 \cdot 2^{n-i} \cdot (1/4) = 3 \cdot 2^{n-i} \cdot 2^{-2} = 3 \cdot 2^{n-i-2} = 3 \cdot 2^{n-(i+2)}$.
Thus $2^{n-i} - 2^{n-(i+2)} = 3 \cdot 2^{n-(i+2)}$ in modulo 13 algorithm. This will be used in Theorem 4.5 to show that modulo 13 algorithm detects all types of jump transpostion errors. □

Theorem 4.4. *Modulo* 13 *algorithm detects all types of transposition of adjacent digits.*

Proof. Let $u = a_1a_2a_3...a_n$ be a $n-$digit code word in which two digits are interchanged, that is, $u = a_1a_2...a_ia_j...a_n$ becoming $v = a_1a_2...a_ja_i...a_n$ where $\forall a \in \mathbb{Z}_{13}$, $a_i \neq a_j$ be digits at position i and j respectively that have been transposed. Hence, the weights at position i and j are 2^{n-i} and 2^{n-j} respectively. The difference $a_i - a_j \neq 0$ and ${}^{13}\nmid_{a_i - a_j}$. Also, since $2^{n-i} > 2^{n-j}$, $2^{n-i} - 2^{n-j} = 2^{n-j}$ as seen in Theorem 4.2. Hence, $\left[\left(a_i - a_j\right)2^{n-j}\right] \ mod\, 13 \neq 0$ since ${}^{13}\nmid_{2^{n-j}}$. Thus, modulo 13 algorithm detects any type of adjacent transposition error. □

An example of this is a modulo 13 algorithm code word 44781500**19**042173 in which two adjacent digits are transposed to obtain 44781500**91**042173.
Note that the transposition occurs at position $i = 10$ and position $j = 11$. The weights at position i and j are therefore 2^6 and 2^5 respectively.
$2^6 - 2^5 = 64 - 32 = 32 = 2^5$, $a_i - a_j = 9 - 1 = 8$ and $8 \cdot 2^5 = 256\,(mod\,13) \neq 0$. Therefore modulo 13 algorithm detects all types of adjacent transposition errors.

Theorem 4.5. *Modulo* 13 *algorithm detects all types of jump transposition errors.*

Proof. Let $u = a_1a_2a_3...a_n$ be a $n-$digit code word that has undergone jump transposition at the i^{th} and k^{th} position, that is , $u = a_1a_2...a_ia_ja_k...a_n$ becoming $v = a_1a_2....a_ka_ja_i...a_n$

where $a_i \neq a_k$ hence, $^{13}\nmid_{a_i - a_k}$. The weights at position i and k are 2^{n-i} and 2^{n-k} respectively, where $2^{n-i} > 2^{n-k}$. The difference $2^{n-i} - 2^{n-k} = 3 \cdot 2^{n-k}$ as seen in Theorem 4.3 and $^{13}\nmid_{3 \cdot 2^{n-k}}$ since $^{13}\nmid_{2^{n-k}}$. This implies that $^{13}\nmid_{(a_i - a_k)3 \cdot 2^{n-k}}$. Therefore $\left[\left(a_i - a_j\right) 3 \cdot 2^{n-k}\right] mod\, 13 \neq 0$. This implies that the jump transposition error will affect the checksum, consequently change the check digit. Thus modulo 13 algorithm detects all types of jump transposition errors. □

Transposition errors will remain undetected if the transposed digits are equal, that is, $a_i = a_k$ for jump transposition, and $a_i = a_j$ for adjacent transposition. In that case there is no error which has occured.

4.4.3 Twin error detection

Theorem 4.6. *Let $n \in \mathbb{N}$ be the number of digits in a code word and $i \in \mathbb{N}$ be the position of a digit, the sum of the weights $2^{n-i} + 2^{n-(i+1)} = 3 \cdot 2^{n-(i+1)}$ in modulo 13 algorithm.*

Proof. In modulo 13 algorithm, the weights used are of the form
$2^{n-1}, 2^{n-2}, 2^{n-3}, \cdots, 2^{n-i}, 2^{n-(i+1)}, \cdots, 2^1, 2^0$.
Thus $2^{n-i} + 2^{n-(i+1)} = 2^{n-i} \cdot \left(2^0 + 2^{-1}\right) = 2^{n-i} \cdot (1 + 1/2) = 2^{n-i} \cdot (3/2) = 3 \cdot 2^{n-i} \cdot 2^{-1} = 3 \cdot 2^{n-(i+1)}$ in modulo 13 algorithm.
Thus $2^{n-i} + 2^{n-(i+1)} = 3 \cdot 2^{n-(i+1)}$ in modulo 13 algorithm. This Theorem will be used in Theorem 4.7 to show that modulo 13 algorithm detects all types of twin errors. □

Theorem 4.7. *Modulo 13 algorithm detects all types of twin errors.*

Proof. Let $u = a_1 a_2 a_3 ... a_n$ be a $n-$digit modulo 13 algorithm code word in which a twin error has occured *i.e.* $u = a_1 a_2 ... a_i a_j ... a_n$ becoming $v = a_1 a_2 ... b_i b_j ... a_n$ where $a_i = a_j$ and $b_i = b_j$ but $a_i \neq b_i$. The weights at position i and j are 2^{n-i} and 2^{n-j} respectively where $2^{n-i} > 2^{n-j}$. Therefore, the checksum difference associated with the digits in error is $a_i \cdot 2^{n-i} + a_j \cdot 2^{n-j} - b_i \cdot 2^{n-i} - b_j \cdot 2^{n-j}$ since $a_i = a_j$ and $b_i = b_j$ we have $a_i \cdot 2^{n-i} + a_i \cdot 2^{n-j} - b_i \cdot 2^{n-i} - b_i \cdot 2^{n-j} = 2^{n-i}(a_i - b_i) + 2^{n-j}(a_i - b_i) = (a_i - b_i) \cdot \left(2^{n-i} + 2^{n-j}\right)$.
$^{13}\nmid_{a_i - b_i}$ and for all i and j , $2^{n-i} + 2^{n-j} = 3 \cdot 2^{n-j}$ from Theorem 4.6. Since $^{13}\nmid_{2^{n-j}}$ then $^{13}\nmid_{3 \cdot 2^{n-j}}$ implying that
$\left[(a_i - b_i) \cdot \left(2^{n-i} + 2^{n-j}\right)\right] mod\, 13 \neq 0$. Hence modulo 13 algorithm detects all types of twin errors. □

4.4.4 Jump twin error detection

Theorem 4.8. *Let $n \in \mathbb{N}$ be the number of digits in a code word and $i \in \mathbb{N}$ be the position of a digit, the sum of the weights $2^{n-i} + 2^{n-(i+2)} = 5 \cdot 2^{n-(i+2)}$ in modulo 13 algortihm.*

Proof. In modulo 13 algorithm, the weights used are of the form
$2^{n-1}, 2^{n-2}, 2^{n-3}, \cdots, 2^{n-i}, 2^{n-(i+1)}, 2^{n-(i+2)}, \cdots, 2^1, 2^0$.
Thus

$$2^{n-i} + 2^{n-(i+2)} = 2^{n-i} \cdot \left(2^0 + 2^{-2}\right) = 2^{n-i} \cdot (1 + 1/4) = 2^{n-i} \cdot (5/4) = 5. \cdot 2^{n-i} \cdot 2^{-2} = 5 \cdot 2^{n-(i+2)}$$

in modulo 13 algorithm. Thus $2^{n-i} + 2^{n-(i+2)} = 5 \cdot 2^{n-(i+2)}$ in modulo 13 algorithm. □

Theorem 4.9. *Modulo* 13 *algorithm detects all types of jump twin errors*

Proof. Let $u = a_1 a_2 \ldots a_i a_j a_k \ldots a_{n-1} a_n$ where $a_i = a_k$ be a $n-$digit codeword in modulo 13 algorithm that undergoes jump twin error to obtain code word $v = a_1 a_2 \ldots b_i a_j b_k \ldots a_{n-1} a_n$ in which digit a_i and digit a_k have been replaced with digits b_i and digit b_k where $b_i = b_k$ and $a_i \neq b_i$. The weights at position i and k are 2^{n-i} and 2^{n-k} respectively. The difference in checksum contribution between codewords u and v is

$$a_i \cdot 2^{n-i} + a_k \cdot 2^{n-k} - b_i \cdot 2^{n-i} - b_k \cdot 2^{n-k},$$

but since $a_i = a_k$ and $b_i = b_k$,

$$\begin{aligned} a_i \cdot 2^{n-i} + a_i \cdot 2^{n-k} - b_i \cdot 2^{n-i} - b_i \cdot 2^{n-k} &= 2^{n-i}(a_i - b_i) + 2^{n-k}(a_i - b_i) \\ &= (a_i - b_i)\left(2^{n-i} + 2^{n-k}\right) \end{aligned}$$

$^{13} \nmid_{a_i - b_i}$ for all $a, b \in \mathbb{Z}_{13}$ and for all i and k , $2^{n-i} + 2^{n-k} = 5 \cdot 2^{n-k}$. Since $^{13} \nmid_{5.2^{n-k}}$ from Theorem 4.8 implyies that $\left[(a_i - b_i) \cdot \left(2^{n-i} + 2^{n-k}\right)\right] \; mod \; 13 \neq 0$.
Therefore, modulo 13 algorithm detects all types of jump twin errors. □

4.4.5 Phonetic error detection

Theorem 4.10. *Modulo* 13 *algorithm detects all phonetic errors.*

Proof. Phonetic errors are errors of the form $1a \longrightarrow a0$. Suppose two adjacent digits $\ldots a0 \ldots$ in a code word which is mistaken for digits $\ldots 1a \ldots$. If the digits are at position i and j in the code word then the weights involved are 2^{n-i} and 2^{n-j} respectively. The checksum difference contributed by the error is
$a \cdot 2^{n-i} - 2^{n-i} - a \cdot 2^{n-j} = (a-1)\,2^{n-i} - a \cdot 2^{n-j}$. Since $\forall\, k \in \mathbb{N}$, $^{13} \nmid_{2^k}$ and $^{13} \nmid_{a-1}$ $\forall a \in \mathbb{Z}_{13}$, then $\left[(a-1)\,2^{n-i} - a \cdot 2^{n-j}\right] mod \; 13 \neq 0$.
Therefore, modulo 13 algorithm detects all phonetic errors. □

4.4.6 Increased dictionary

Luhn algorithm or modulo 10 formula has a dictionary size of 10^{15} in credit card numbers while modulo 13 algorithm has dictionary size of 13^{15}in credit card numbers. Hence the increase is $13^{15} - 10^{15} = 5.019 \times 10^{16}$. Thus, there is a tremendous increase of credit card numbers for the rapidly increasing human population.

CHAPTER FIVE
CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusion

As observed in Chapter 2 and Chapter 3 Luhn algorithm is not effective in error detection and correction. Usage of a composite modulo, that is modulo 10 and modulo 9 in error detection allows errors associated with zero divisors to be undetected. This contributes to the reason why the Luhn formula does not detect some twin errors and transposition of digits 09 for 90. From the research, it is observed that adding digits of the products $p > 9$ is the same as calculating $p \ mod \ 9$ if $9 \nmid p$ or $p-9$ if $9 | p$. As a result, Luhn does not solve the issue of a composite modulus. Hence, Luhn algorithm can't detect errors associated with zero divisors and that is why some of the errors still pass undetected. In order to detect twin errors $22 \longrightarrow 55$, $33 \longrightarrow 66$, $44 \longrightarrow 77$ and transposition error $09 \longrightarrow 90$, guidelines on how to choose the codeword digits have been provided for effective error detection. Usage of alternating weights $(2,1,2,1,...)$ in the Luhn algorithm makes it not to detect any type of jump transposition. These weaknesses necessitated the creation of a new algorithm, the modulo 13 algorithm, that surpasses the Luhn algorithm in error detection. Modulo 13 algorithm uses a prime modulus, 13, which captures all errors associated with zero divisors. Also it uses weights in the form of decreasing geometric progression of 2^{n-i} that ensures all jump transposition errors are captured. Also, the modulo 13 algorithm increases the dictionary of the codewords tremondously.

5.2 Recommendations

Having achieved the objectives of the current study, these are other interesting areas of this study that have not received attention. Thus studies should be done in the following enumerated areas:

1. Design of an automated conversion tool for the existing Luhn algorithm to modulo 13 algorithm.

2. Create an error correcting formula for the modulo 13 algorithm.

3. Analyse the effectiveness of using weights p^{n-1} in form of a decreasing geometric progression where p is prime and when p is composite in a checksum algorithm.

REFERENCES

Bowman, J. (2010). *Coding theory and cryptography*. Edmonton: University of Alberta.

Causley, B. (2012). The secret behind the Luhn-ie. *XRDS*, *19*(1), 1 – 2.

Celko, J. (2010). *Joe Celko's data measurements and standards in SQL*. Mor-gan Kaufmann Publishers.

Child's, L. (1979). *A concrete introduction to higher algebra.* Newyork: Springer Science and BusinessMedia.

Gallian, J. (1991). The mathematics of identif cation numbers. *The College of Mathematics Journal*, *22*(3), 194 – 202.

Ghandhi, K. (2015). *Check digits*. HTTP://cosmos.ucdavis.edu/archives/2010/cluster6/ghandhikira.HTML. 6/8/2015.

Gupta, V., & Verman, C. (2012). Error detection and correction: An introduction. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(11), 212 – 218.

Hodge, J., Schlicker, S., & Sundstrom, T. (2013). *Abstract algebra: An inquiry-based approach*. Boca Raton: Chapman and Hall/ CRC Press.

Hussein, K. (2013). Enhanced luhn algorithm for validation of credit card numbers. *International Journal of Computer Science and Mobile Computing*, *2*(7), 262 – 272.

Kamaku, W.(2012). *Error detection and correction on the international standard book number.* Unpublished PHD thesis: Jomo Kenyatta University of Agriculture and Technology.

Kumar, K., & Kaur, P. (2015). Vulnerability detection of international mobile equipment identity number of smartphone and automated reporting of changed IMEI number. *International Journal of Computer Science and Mobile Computing*, *4*(5), 527 – 533.

Mohr, J. (2015). *Check digits*. HTTP://www.augustana.ab.ca/mohrj/algorithms/checkdigit.HTML. 7/10/2015.

Rane, H. (2015). Error detection in numeric code. *International Journal of Computer Science and Information Technologies*, *6*(1), 781 – 784.

Raymond, H. (1986). *A f rst course in coding theory.* Oxford: Clarendon Press.

Rosen, K. (2012). *Discrete mathematics and its applications*. New York: Oxford University Press.

Sutherland, D. (1990). Error-detecting identif cation code for algebra students. *School Science and Mathematics*, *90*(4), 283 – 290.

Tervor, R. (2014). *Secrets of Luhn-10 algorithm: An error detection method*. HTTP://www.ee.unb.ca/tervo/ee4253/luhns.HTML. 7/10/2015.

Tilborg, H. (1993). *Coding theory*. Netherlands: Eindhoven.

Vasiga, T. (2008). *Error detection in number- theoretic and algebraic algorithms*. Unpublished PHD thesis: University of Waterloo.

Weisstein, E. (2015). *Error correcting code*. HTTP://www.mathworld.wolfram.com /Error-Correcting Code. HTML. 7/10/2015

APPENDIX

PUBLICATION

Wachira, W., Kamaku, W and Lewis, N. (2015). Transposition Error Detection in Luhn's Algorithm. *International Journal of Pure and Applied Sciences and Technology*, 30(1) : 24 – 28.