UNIVERSITY FOR DEVELOPMENT STUDIES

AN EFFICIENT ALGORITHM FOR RESIDUE NUMBER SYSTEM IMPLEMENTATION OF RIVEST, SHAMIR AND ADLEMAN (RSA) CRYPTOGRAPHY

BY

ISMAIL RASHID FADULILAHI

(BSc. Financial Mathematics)

(UDS/MM/0011/12)

THESIS SUBMITTED TO THE DEPARTMENT OF MATHEMATICS, FACULTY OF MATHEMATICAL SCIENCES, UNIVERSITY FOR DEVELOPMENT STUDIES IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE DEGREE IN COMPUTATIONAL MATHEMATICS

JULY, 2015

DECLARATION

Student

I hereby declare that this thesis is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere except the references to other researchers or writers which have been duly acknowledged.

Candidate's Signature..... Date 01/11/15

Name: **Ismail Rashid Fadulilahi**

Supervisor

I hereby declare that the preparation and presentation of this thesis was supervised in accordance with the guidelines on supervision of thesis laid down by the University for Development Studies.

Supervisor's Signature..... Date 05/11/15

Name: **Dr. Edem K. Bankas**

ABSTRACT

In this thesis, an algorithm for Residue Number System (RNS) implementation of Rivest Shamir and Adleman (RSA) cryptography based on an existing RNS division algorithm is proposed. Many RNS division algorithms have been proposed over the years targeted at application to areas like cryptography. However all of these algorithms are restricted either due to the number of iterations involved or in terms of generality and the type of quotients involved. Iterations in RNS division algorithms mostly lead to long execution time, large hardware requirements, high cost for implementation, and high power consumption among others. However, performing division in RNS using a non-iterative and pure RNS division algorithm avoids iterations, high hardware requirements, high cost implementation, restrictions from the type of quotient and high time consumption. The proposed algorithm and that of the state of the art were written in C++ programming language to compare their efficiency with respect to execution time using two different moduli sets with dynamic range of 29 and 692 respectively. The study reveals that show that our proposed algorithm can encrypt and decrypt text without loss of inherent information and faster than the state of the art. It also offers firm resistance to Brute-force and key sensitivity attacks. We carried out an error analysis of the experimental results at 95 degrees significance level. The statistics showed a low level of error in our algorithm.

ACKNOWLEDGEMENT

First and foremost, my deepest appreciation goes to my supervisor, Dr. Edem K. Bankas, for his patience, encouragement, guidance and immense support towards the success of this thesis.

My second sincerest appreciation goes to Professor Kazeem Alagbe Gbolagade for his care, guidance and all the contribution and support he gave me. I wish to also thank Mr. Daabo I. M. for his brotherly care, encouragements, guidance and contributions he made to me towards the success of this work.

In addition, I sincerely appreciate the joy, happiness and fun with my colleagues, Mr Siedu Azizu, Madam Rafiatu Imoro, Mr. Arthur E. Maurice, Mr Christian E. John and Mr John Bosco Ansuura. I am also grateful to my family for their support especially my beloved Ida.

I also wish to thank Mr. Abdul Ghaniyyu Abubakari of African Institute for Mathematical Sciences-Ghana, Mr. Jonas Baba and Mr. Aveyom Otto, a special thanks to you for your sleepless nights and efforts for the success of this project.

Finally the greatest of my appreciation goes to the Almighty Allah for the abundance of blessings upon my life.

DEDICATION

Which is it of the favours of the lord that ye deny? God is love. With love this thesis is dedicated to my beloved Father the late Mr. Mohammed Ismail Rashid of blessed memory.

TABLE OF CONTENTS

LIST OF TABLES

List of Acronyms

BM	Bisection Method
CRT	Chinese Remainder Theorem
DES	Data Encryption Standard
DSA	Digital Signature Algorithm
GCD	Greatest Common Divisor
HPT	Highest Powers of Two
MRC	Mixed Radix Conversion
MM	Montgomery Multiplication
OS	Operating System
PCT	Parity Checking Technique
RSA	Rivest-Shamir-Adleman
RNS	Residue Number System
WNS	Weighted Number System

CHAPTER ONE

INTRODUCTION

1.1 Introduction and Background of the Study

In recent times, there has been a vigorous and continuous search for improving computer performance Parhami (1999). Researchers are coming out with new ideas and technologies to make the computer more efficient. The main task of a computer is computing which deals with numbers all the time. Some examples of number systems are binary number systems, decimal number systems, Weighted Number System (WNS) and Residue Number System (RNS). Binary and decimal number systems, intrinsically limit the performance of arithmetic units and processors built based on them. Because of these limitations in Weighted Number System (WNS), RNS has the following advantages of computing large numbers over WNS. These include carry-free addition and borrow free subtraction, which are the challenges to binary and decimal number system because, in RNS a number is represented by the residues of all moduli, and the arithmetic can be performed on each modulus independently. Therefore RNS offers the properties of parallelism (Szabo, 1967).

Even though RNS has many advantages over WNS in terms of encoding large numbers into a set of smaller numbers to speed up computations the following are time-consuming operations in RNS which affect the wide spread application of RNS, in areas like cryptography, overflow detection, sign detection, magnitude comparison and division. Among them, division has modular operations application as can be found in cryptography (Szabo, 1967).

Currently fast hardware implementations of RSA cryptosystem is under study while confidentiality and security requirements are becoming more and more important. In view of this emerging problem of digital security, cryptographers keep increasing the key-length.

Recently, it is assumed that a 1024-bit key-length makes a reasonable choice for the cryptography proposed by Rivest et al., (1978), popularly known as RSA, and current analysis predict that 2048-bit or 4096-bit key will become the standard in a near future (Rivest et al., 1978).

The ability to perform fast arithmetic on large integers is still a major issue for the implementation of public key cryptography and digital signature, particularly from the hardware design point of view (Rivest et al., 1978).

1.2 Division Algorithms and Executing Time in RNS

A lot of division algorithms have been developed including the general division method over the years. However, all these proposed algorithms require long execution time and large hardware resources. This is attributed to the methods used, namely the Mixed-Radix Conversion (MRC), Montgomery Multiplication (MM) and Chinese Remainder Theorem (CRT) (Hiasat and Zohdy, 1997).

Instead of employing MRC and CRT, Hiasat and Zohdy (1997) used the highest powers two comparison between the dividend and the divisor to design a division algorithm in RNS. Their algorithm computes the evaluated quotient according to the highest powers of the dividend and the divisor, and obtains the actual quotient by computing the sum of all evaluated quotients until the product of the divisor and the evaluated quotient is less than the dividend. However, the evaluated quotient is underestimated in their algorithm. Thus, Yang et al., (2004) proposed a division algorithm in RNS using the parity checking technique. In the algorithm, the evaluated quotient is estimated precisely such that approximating the actual quotient by adding the evaluated quotients is two times faster than that in the algorithm proposed by (Hiasat and Zohdy, 1997). However, computing the highest power of 2 of a number is time-consuming in RNS (Yang et al., 2004).

Chang and Yang, (2013) designed a division algorithm, which limits the above constrains using the bisection method in RNS. It is a new algorithm which finds the quotient in a probable interval efficiently. The algorithm only needs to compute the highest-power comparison in RNS in the first round.

1.3 Basics of Cryptography

Cryptography is a method of turning readable file into gibberish. We can use cryptography to store, transmit or receive sensitive data across insecure networks like (the internet) so that it cannot be read by anyone except the recipient (Steve and Stephen, 2004).

Plain-text is data which can be read and understood without any special measures. Cipher-text is when plain-text is disguised in such a way as to hide its substance. Encryption is used to get cipher-text and decryption is also used to get plain-text back (Steve and Stephen, 2004).

1.3.1 THE ROLE OF CRYPTOGRAPHY

Most often than not we have secrets that need protection, these secrets appear in areas like medical files, bank statements, paycheck, investment portfolio and credit card bills. Others include social security numbers, credit card numbers, bank account numbers and the PIN for ATM credit card or phone card among others. We all have information we want to keep private.

Corporations also have secrets, strategy report, sales forecasts, technical product details, research results and personnel files among others

In the past before the advent of computers, security was simply a matter of locking of doors or storing files in locked filling cabinet or safe. Today files are stored in computer databases as well as file cabinets.

Hard-drives and floppy disks hold many of our secrets. In the beginning the best way was to provide security to these data through the Operating System (OS), by locking it using a password.

However, various attacks on passwords have rendered this mode of security vulnerable and attacks by pass the OS. For your secrets to be secured, it may be necessary to add protection not provided by a computer (OS).One of the most important tools for protecting data is cryptography (Steve and Stephen, 2004).

1.3.2 ADVANTAGES OF CRYPTOGRAPHY

Some advantages of cryptography include;-

- Adding security to the process of authenticating people identity.
- Improves privacy, meaning that, no one can break into files to read your sensitive data.
- Another concept is data integrity, which refers to a mechanism that tells us when something has been altered.
- By applying the practice of authentication, you can verify identities.
- Non repudiation, a legal driving force that impels people to honor their word is also used in cryptography (Steve and Stephen, 2004).

1.3.3 COMMON CRYPTOGRAPHIC ALGORITHMS

Data Encryption Standard (DES) is the most popular computer encryption algorithm. DES is of US and international standard. It is a symmetric algorithm, that is, the same key is used for encryption and decryption (Steve and Stephen, 2004).

The algorithm proposed by Rivest et al. (1978), known as RSA, is a public key algorithm. It can be used for both encryption and digital signatures (Rivest et al., 1978).

Digital Signature Algorithm (DSA) is another public key algorithm. It cannot be used for encryption, but can only be used for digital signatures (Rivest et al.,1978)

1.3.4 CRYPTANALYSIS

Cryptanalysis is the science of analyzing and breaking through secure communication. This involves combination of critical thinking, application of mathematical tools, creating patterns. Cryptanalysts can be referred to as attackers. In cryptography, the secrecy must reside entirely in the key Rivest et al. (1978)

1.3.5 COMMON CRYPTANALYTIC ATTACKS

The cryptanalytic attacks presented here are all assumed, the cryptanalyst has full knowledge of the encryption algorithm used. These include;

i. Cipher-text-only attack. The cryptanalyst has the cipher-text of several massages, all of which have been encrypted using the same encryption algorithm. The job of the Cryptanalyst is to recover the plain-text of as many messages as possible, or better yet to deduce the key or keys used to encrypt the messages in order to decrypt other encrypted with the same keys.

ii. Known-plain-text attack. The cryptanalyst has access not only to the cipher-text of several messages, but also to the plain-text of those messages. His job is to deduce the key or keys used to encrypt the messages or an algorithm to decrypt any new messages encrypted with the same key or keys.

iii. Chosen-plain-text attack. The cryptanalyst not only has access to the cipher-text and associated plain-text for several messages, but he also chooses the plain-text that gets encrypted. This is more powerful than a known-plain-text attack, because the cryptanalyst can choose specific plain-text blocks to encrypt, ones that might yield more information about the key. His job is to deduce the key or keys used to encrypt

the messages or an algorithm to decrypt any new messages encrypted with the same key or keys.

iv. Adaptive-chosen-plain-text attack. It is a special case of a chosen-plain-text attack. Not only can the cryptanalyst choose the plain-text that is encrypted, but he can also modify his choice based on the results of previous encryption. In the chosen-plain-text attack, a cryptanalyst might just be able to choose one large block of plain-text to be encrypted, in an adoptive-chosen-plain-text attack he can choose a smaller block of plain-text and then choose another based on the results of the first, and so forth.

v. Chosen-cipher-text attack. The cryptanalyst can choose different cipher-texts to be decrypted and has access to the decrypted plain-text.

vi. Chose-key attack. The cryptanalyst has some knowledge about the relationship between keys (Alhassan and Gbolagade, 2013)

1.4 Problem Statement

Many Algorithms for division in RNS are presented in international journals by Parhami (1994), Yang et al., (2004) Hiasat and Zohdy (1997). However, most of these algorithms are iterative and has the problem of an overall loop and not supporting all numbers in the dynamic range as denominator which affects their application to other fields like RSA cryptography. This study proposed an efficient RNS implementation of RSA cryptography based on a non-iterative and pure RNS division algorithm by Mansoureh and Mohammed (2012), which avoids overall loop and support all numbers in the range as denominator.

1.5 Objectives of the Study

1.5.1 GENERAL OBJECTIVE OF THE STUDY

The main objective of this study is to use a non-iterative and pure RNS division algorithm by Mansoureh and Mohammed (2012) to design an algorithm that can be applied to simplify and improve the security in RSA cryptography

1.5.2 SPECIFIC OBJECTIVES

The specific objectives of the study are;

i. To reduce computational time and enhance the efficiency in RNS operations involving division and applying it to RSA public key cryptography
ii. To absorb or control the emerging problem of growing key length in RSA
iii. To propose a general algorithm that can be applied to RSA cryptography based on the division algorithm such that fractions, decimals, integers can all be encrypted and decrypted.
iv. To improve on existing data encryption techniques by developing an efficient algorithm.

1.6 Research Questions

i. Can we apply the division algorithm based on a non-iterative and pure RNS division algorithm by Mansoureh and Mohammed (2012) to simplify and improve the security of RSA public key cryptography?
ii. Will the chosen algorithm be able to absorb or control the emerging problem of growing key length in RSA?
iii. Can we achieve a general algorithm that can be applied to RSA cryptography based on the division algorithm using a non-iterative and pure RNS division algorithm by

Mansoureh and Mohammed (2012)in RNS such that fractions, decimals, integers can all be encrypted and decrypted?

iv. Will the proposed algorithm performed relatively well comparing it to existing algorithm?

1.7 Significance of the Study

RSA cryptosystem is most commonly used for providing privacy and ensuring authenticity of digital data. These days, RSA is found in many commercial systems. Such as-:

- Web servers and browsers used it to secure web traffic, ensure privacy and authenticity of E-mail, to secure remote login sessions.
- Electronic credit-card payment systems.

In summary, RSA is frequently used in applications where security of digital data is concern. With the increasing use of digital techniques for transmitting and storing information, the fundamental issue of protecting the confidentiality, integrity as well as the authenticity of cryptosystem has become a major concern. Various data protection techniques have been proposed over the years to solve these problems. The significance of this research is to improve on existing data encryption techniques by developing an efficient algorithm.

1.8 Definition of Terms

Algorithm: This is a scientific term for a recipe or step by step procedure. It is a list of instructions or things to do in a particular order. An algorithm might have a rigid list of commands or it might contain a series of questions and depending on the answers describe the appropriate steps to follow. A mathematical algorithm might list the operations to perform in a particular order to find a variable. Algorithm– a set of mathematical instructions that

must be followed in a fixed order, and that, especially, if given to a computer, will help to calculate an answer to a mathematical problem (Steve and Stephen, 2004).

Cryptography: The art and science of keeping messages secure is cryptography, and it is practiced by which readable text are turn to gibberish text to prevent anyone from accessing to it apart from the one it is intended for.

Cryptanalysts are practitioners of cryptanalysis, the art and science of breaking cipher-text; that is, seeing through the disguise (Steve and Stephen, 2004).

Encryption and Decryption: A message is plaintext (sometimes called clear text). The process of disguising a message in such a way as to hide its substance is encryption. An encrypted message is cipher-text. The process of turning cipher text back into plaintext is decryption (Steve and Stephen, 2004).

Cryptology and Cryptologists: The branch of Mathematics encompassing both cryptography and cryptanalysis is cryptology and its practitioners are cryptologists.

Key: This is a number or set of numbers which can serve as a password or code. Key is simply a number that is used to encrypt or decrypt data (Steve and Stephen 2004).

1.9 Organization of the study

This thesis composed of five chapters. All the chapters are based on the thesis topic, an efficient algorithm for RNS implementation of RSA. Lectures and other related works from other publishers in the field of RNS division algorithms and their application to RSA algorithms.

Chapter one presents the introduction, objectives, research questions and significances of the study. Chapter two deals with the literature review of RNS division algorithms and RSA

algorithms and how RNS can be applied to RSA. Chapter three presents the methodology and the proposed algorithm. Chapter four presents the discussion and analysis of the results. Security analysis, key sensitivity analysis among others were discussed. Chapter five deals with the conclusion, summary, recommendations and future research directions.

CHAPTER TWO

LITERATURE REVIEW

2.1 The Concept of RNS and Cryptography

Splitting a large number into a group of small numbers will results in significant computational speed (Parhami, 1994). Division in RNS is of much interest because it is one of the four basic mathematical operations. Number theory may be one of the "purest" branches of Mathematics; it has turned out to be one of the most useful branches when it comes to computer security. For instance, number theory helps to protect sensitive data such as credit card numbers when you shop online. This is the result of some remarkable mathematics research from the 1970s that is now being applied worldwide (Parhami, 1999).

Sensitive data exchanged between a user and a Web site needs to be encrypted to prevent it from being disclosed to or modified by unauthorized parties.

In traditional cryptography, encryption and decryption operations are performed with the same key, that is, symmetric key cryptography. This means that the party encrypting the data and the party decrypting it need to share the same decryption key. If two parties already share a secret key, they could easily distribute new keys to each other by encrypting them with prior keys. From symmetric key encryption, researchers continue to build knowledge towards unsymmetrical key encryption.

Diffie and Hellman (1976) suggested that, encryption and decryption could be done with a pair of different keys. The decryption key would be kept secret, and the encryption key could be made public. This concept was called public-key cryptography.

Every computer can use that encryption key to protect data sent to the site. However, only the site has the corresponding decryption key that can decrypt the data.

Diffie and Hellman (1976) introduced the concept of digital signatures.

Their key agreement is confronted with the problem of discrete logarithms and integer factorization. In 1977, a public key cryptosystem was invented by Rivest et al. (1977), called the RSA public key cryptography. However, only integers are encrypted in RSA. Hung and Parhami (1994) first wrote on fast RNS division algorithms for fixed divisors with application to RSA encryption. Their algorithm was restricted to only division with integer quotient. It was having longer time consumption because of its iterative.

Imbert and Bajard (2004) proposed a full RNS implementation which was based on the CRT, MM, and base extension. Gbolagade (2010) mentioned that the main drawback of CRT emerges from the required modulo-M operation, which given that M is a rather large number, this operation can be time consuming and rather expensive in terms of area and energy consumption. Chang and Yang (2013) proposed a division algorithm without using CRT/MRC or MM. The authors used the parity checking technique and highest power of two's to perform division in RNS, however it is also iterative. A non-iterative and pure RNS division algorithm by Mansoureh and Mohammed (2012) was also proposed and this solved the looping problem and restriction to integer quotient.

2.2 Overview of Residue Number System (RNS)

Residual Number System (RNS) were invented by a third-century Chinese scholar Sun Tzu (Omondi et al., 2007). RNS is an integer system which speeds up arithmetic computations by splitting numbers into smaller parts in such a way that each part is independent of the other (Gbolagade et al., 2009).

RNS is suited for the implementation of fast arithmetic and fault-tolerant computing due to the following inherent properties-;

- Absence of carry-propagation in addition and multiplication
- Residual representation carry no weight-information

- There is no significance-ordering of digits in an RNS representation.

These inherent features make RNS to be widely used in Digital Signal Processing (DSP) applications such as digital filtering, convolution, cryptography etc. (Gbolagade et al., 2009; Pemmaraj, 2009). The application of RNS in cryptography and image processing is been paid much attention to by researchers (Weyori et al., 2012; Shahram et al., 2012;Talesshmeka et al., 2012). The aims of such applications are to ensure fast data transmission, conversion of disk space and optimization of internal memory (Weyori et al., 2012; Shahram et al., 2012; Talesshmekael et al., 2012).

2.2.1 FUNDAMENTALS OF RESIDUE NUMBER SYSTEM (RNS)

RNS is defined by a set of relatively prime integers called the moduli. The moduli-set is denoted by $\{m_1, m_2, ..., m_n\}$ where m_i is the i^{th} modulus. Each integer X can be represented as a set of smaller integers called the residues. The residue-set denoted as $\{x_1, x_2, ..., x_n\}$ is the residue where x_i is the i^{th} residue. The residue x_i is defined as the least positive remainder when X is divided by the modulus m_i. The notation for this relation can be written based on the congruence of the form:

$$mod m_{i=X_I} \quad (2.1)$$

The same congruence can be written in an alternative notation as

$$|X|_M = m_i \quad (2.2)$$

The RNS is capable of uniquely representing all integers X that lie in its dynamic range. The dynamic range is determined by the moduli-set $\{m_1, m_2, ..., m_n\}$ and denoted as M where:

$$M = \prod_{i=1}^{n} m_i \quad (2.3)$$

The RNS provides unique representation for all integers in the range between 0 and M-1. If the integer is greater than M-1, the RNS representation repeats itself. Therefore, more than one integer might have the same residue representation.

It is important to emphasize that the moduli have to be relatively prime to be able to exploit the full dynamic range (Abdelfattah, 2011).

Let S denote the moduli set, then

$S = \{ m_1, m_2 \} = \{3, 7\}$ and $M = 3 \times 7 = 21$, $GCD = (m_i, m_j) = 1$, for $i \neq j$

Illustration to Show That RNS has a Unique Representation

Consider two different residue number systems defined by the two moduli-sets {2,3,5} and {2,3,4}. The representation of the numbers in residue format are as shown in Table 1.1.

Table 2.1 Unique Representations of RNS

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

Source, Abdelfattah (2011)

In the first RNS, the moduli-set {2, 3, 5} are relatively prime. The RNS representation is unique for all numbers in the range from 0 to 29. Beyond that range, the RNS representation repeats itself. For example, the RNS representation of 30 is the same as that of 0.

In the second RNS, the moduli-set {2, 3, 4} are not relatively prime, since 2 and 4 have a common divisor of 2. We see that the RNS representation repeats itself at 12 preventing the dynamic range from being fully exploited. Therefore, choosing relatively prime moduli for the RNS is necessary to ensure unique representation within the dynamic range.

In the preceding discussion on RNS, we assumed dealing with unsigned numbers. However, some applications require representing negative numbers. To achieve that, we can partition the full range [0: M-1] into two approximately equal halves: the upper half represents the positive numbers, and the lower half represents the negative numbers. The numbers X that can be represented using the new convention have to satisfy the following relations:

$[-\frac{M}{2}, \frac{M}{2} - 1]$, If M is even

$[\frac{(M-1)}{2}, \frac{(M+1)}{2}]$ if M is odd

2.2.2 ARITHMETIC OPERATION IN RNS DOMAIN

Addition and subtraction

Congruence with respect to the same modulus may be added or subtracted, and the result will be a valid congruence. That is, if

$$X = x(\mathrm{mod}\, m)$$

$$Y = y(\mathrm{mod}\, m)$$

then

$$X \pm Y \equiv x \pm y(\mathrm{mod}\, m)$$

Multiplication

Congruence with respect to the same modulus may be multiplied, and the result is a valid congruence. That is, if

$$X = x(\text{mod}\, m)$$

$$Y = y(\text{mod}\, m)$$

then

$$X \times Y \equiv x \times y(\text{mod}\, m)$$

Extension of sum and product properties

The properties above for addition and multiplication have two direct extensions. If $\{x_1, x_2 ..., x_n\}$ and $\{y_1, y_2 ..., y_n\}$ are, respectively the residue sets (without any restrictions) of X and Y, obtained relative to the moduli $m_1, m_2, ... m_n$ then, the residue set of

$$X \pm Y = \{x_1 \pm y_1, x_2 \pm y_2, ..., x_n \pm y_n\}$$

And that

$$X \times Y = \{x_1 \times y_1, x_2 \times y_2, ..., x_n \times y_n\}$$

Example 2.1

Suppose X=21, Y=11 and the moduli set is {2, 3, 5} then

$$X + Y \cong \langle |x_{1+} y_1|_2, |x_2 + y_2|_3, |x_3 + y_3|_5 \rangle$$

Where

$x_i = |X|_{m_i}$

and

$y_i = |Y|_{m_i}$

and

$X = 21$

$Y = 11$

Then

$21 + 11 \cong \langle |1+1|_2, |0+2|_3, |1+1|_5 \rangle$

$32 \cong \langle 0,2,2 \rangle$

2.3 The Chinese Remainder Theorem (CRT)

The Chinese Remainder Theorem (CRT) may rightly be viewed as one of the most important fundamental results in the theory of residue number systems. It is, for example, what assures us that if the moduli of a RNS are chosen appropriately then each number in the dynamic range will have a unique representation in RNS and that from such a representation we can determine the number represented. The CRT is useful in reverse conversion as well as several other operations. Given a set of pair-wise relatively prime moduli, $m_1, m_2, m_3 \ldots, m_n$ and a residue representation ($x_1, x_2, \ldots, x_n$) in that system of some number X,

That is $x_i = |X|_{mi}$, that number and its residues are related by the equation

$$X = \left| \sum^{n}{}_{i=1} x_i \left| M^{-1}{}_i \right|_{m_i} M_i \right|_M \qquad (2.4)$$

The expression above is the Chinese Remainder Theorem (Omondi and Premkumar, 2007; Daabo and Gbolagade, 2012;Gbolagade and Cotofana 2009).

Where

$$M_i = {}^{M}/_{m_i} \qquad (2.5)$$

M= dynamic range and m_i= moduli set

Example 2.2:

Consider the moduli set {3, 5, 7}, and suppose we wish to find the X whose residue representation is {1, 2, 3}.

Solution

M = 3×5×7

$M_1 = {}^{M}/_{m_1}$

$M_1 = (3\times5\times7)/3$

$M_1 = 35$, $M_2 = 21$ and $M_3 = 15$

Where

$\left| M_1 \; M_1^{-1} \right|_3 = 1$

$M_1^{-1} = 2$, $M_2^{-1} = 1$ and $M_3^{-1} = 1$

Then by the CRT, we have (M = 3×5×7) = 105

X = $\left| \sum_{i=1}^{3} x_i X_i \right|_{105}$, therefore; X = $\left| 1\times35\times2 + 2\times21\times1+3\times15\times1 \right|_{105}$ i.e. X = 52.

2.4 Mixed Radix Conversion (MRC)

Given a set of pair-wise relatively prime moduli $\{m_1, m_2, ..., m_n\}$ and a residue representation $\{x_1, x_2, ..., x_n\}$ in that system of some number X, where $x_i = |X|_{mi}$. The number X can be represented uniquely in mixed-radix form as $X = \{z_1, z_2 ..., z_n\}$ where

$$X = z_1 + z_2 m_1 + z_3 m_2 m_1 + \ldots + z_n m_{n-1} m_{n-2} \ldots m_1 \tag{2.6}$$

And

$$0 \le z_i \le x_i$$

The Mixed-Radix Conversion (MRC) establishes an association between the un-weighted, non-positional RNS and a weighted positional mixed-radix system. All what is required is to perform the reverse conversion to obtain the values z_i (Omondi and Premkumar 2007),

The z values are obtained as follows:

$$\begin{aligned} z_1 &= x_1 \\ z_2 &= \left| \left| m_1^{-1} \right|_{m_2} (x_2 - z_1) \right|_{m_2} \\ z_3 &= \left| \left| m_2^{-1} \right|_{m_3} \left(\left| m_1^{-1} \right|_{m_3} (x_3 - z_1) - z_2 \right) \right|_{m_3} \\ &\vdots \\ z_N &= \left| \left| m_{N-1}^{-1} \right|_{mN} \left(\left| m_{N-2}^{-1} \right|_{mN} \left(\ldots \left| m_2^{-1} \right|_{mN} \left(\left| m_1^{-1} \right|_{mN} (x_N - z_1) - z_2 \right) \ldots \right) - z_{N-1} \right) \right|_{mN} \end{aligned} \tag{2.7}$$

Example 2.3:

Suppose we wish to find the number, X, whose residue representation is

{1, 0, 4, 0} relative to the moduli set {2, 3, 5, 7}

From the equations above,

$X \cong (1,1,1,6)$

And for the conventional form, we translate this as

$$X = 6\times2\times3\times5+1\times2\times3+1\times2+1$$
$$= 189$$

2.5 Review of Some Division Algorithm in RNS

Most division algorithms in RNS are dependent on either the CRT/MRC starting from the general division. Some of these are discussed below looking forward to constrain and advantages regarding their application to RSA.

2.5.1 GENERAL DIVISION IN RNS

Division is one of the main obstacles that discourage the use of RNS, in RNS representation, division is not a simple operation. The analogy between division in conventional representation and RNS representation does not hold. In conventional representation, we represent division as follows:

$$\frac{x}{y} = q \qquad (2.8)$$

This can be written as

$$y * q = x \qquad (2.9)$$

Where q, is the quotient

In RNS, the congruence:

$$y * q = x mod(m) \qquad (2.10)$$

Multiplying both sides by the multiplicative inverse of y, we can write:

$$q = x * y^{-1} mod(m) \qquad (2.11)$$

The equation $\frac{x}{y} = q$ is equivalent to $q = x \times y^{-1} \bmod m$ only if it has an integer value. Otherwise, multiplying by the multiplicative inverse in RNS representation will not be equivalent to division in conventional representation

Example 2.4

Consider an RNS with m =7, we want to compute the following quotient:

$(i)\ \frac{6}{2}$ $(ii)\ \frac{6}{4}$

In the first case (i)

$$\frac{6}{2} = q = 6 \times 2^{-1}(mod7) = 3$$

This is equivalent to division in conventional representation.

We notice in part (i), that division in RNS is not equivalent to that in conventional representation when the quotient is a non-integer value. Due to this fact, division in RNS is usually done by converting the residues to conventional representation, performing the division, and then converting back to RNS representation. Tedious and complex conversion steps result in undesired overhead. This is one of the main drawbacks of RNS representation.

$$\frac{6}{4} = q = 6 \times 4^{-1}(mod7) = 5$$

However, in cryptography this could serve as an advantage to adding security to cryptosystem.

2.5.2 OTHER DIVISION ALGORITHMS

Many algorithms for division in RNS are presented. Some of these iterative algorithms work by subtracting denominator from numerator in a major loop, until numerator gets less than denominator. Quotient is equal to the number of iterations of this major loop. Some of them

use Newton iteration to compute reciprocal and then compute quotient (Hitz and kaltofen, 1995; Husien et al., 1998).

Another common way for division is using the definition of division. In this algorithm, first the position of the most significant non-zero bit in the divisor and dividend is determined.

Then, according to difference between these two positions, divisor is shifted to the left and is subtracted from dividend. These actions are repeated in a major loop until the result is smaller than divisor. In some methods for dividing X to Y, first the proper $2k$ is detected such that $Y.2k \leq X \leq Y.2k+1$. In the next iterations, these two margins varied until quotient is obtained (Mi Lu, 1992).

There are other methods in which, instead of dividing two proposed numbers, X and Y, two different numbers which have the same ratio and are less than X and Y, are chosen. For doing this, some new moduli are introduced, and at last, in a major loop, the division result is calculated (Berger, 1991; Chang et al., 2004)

These algorithms have three major deficiencies.

i. All of them have an overall loop which increases the complexity and delay of the algorithm.
ii. Some of these methods exclude some numbers in the range of acceptable inputs as a denominator in division operation.
iii. The authors have some operations in the binary or mixed radix system or use a lookup Table to perform an RNS division.

In order to solve problems (i) and (ii) stated above under section (2.5.2), a non-iterative division algorithm was proposed by Mansoureh and Mohammad (2012). Another method of

division algorithm which could be efficient in the implementation of RSA using RNS is the scaling method an integer division.

2.6 Iterative Division Algorithms

These algorithms are iterative in nature and the authors are dependent on the MM, CRT and MRC. Some of them include; the scaling method and Newton's method

2.6.1 THE SCALING METHOD

The first scaling scheme has been proposed by Szabo and Tanaka in 1967. The authors proposed a scaler that needed n clock cycles for n-bit moduli set. Although scaled residues had errors, and the scheme did not provide correct scaled residues, it was a significant stage in the development of RNS-based systems Szabo and Tanaka in 1967. In another major study in 1973, Okeefe and Wright (1973) designed a faster and more efficient scaler than the Szabo scaler. Again the results were not error-free but their approach provided results closer to the correct scaled integers. In 1987 Jullien was successful in designing an algorithm that needed fewer clock cycles, but provided faulty results Jullien 1987. In 1981, Taylor and Huang proposed a design based on the MRC. It was the first time a scaler based on the MRC was proposed. Until then, all designs were based on CRT or base-extension. The CRT-based algorithms generally generated fractional errors due to coarse assumptions, while the latter approach was error-free but computationally intensive. One year later, Taylor and Huang presented a scaler that used a special moduli set and LUTs. However, their design required n clock cycles to generate the scaled residues. In 1984, Polky and Miller proposed a design that needed (n+1) clock cycles but the scaled residues were closer to the correct results. In other words, their design provided more accurate scaled residues at the cost of more clock cycles.

Five years later, Shenoy and Kumarson proposed two scaling techniques (approximate and exact), where residues were scaled by the product of a subset of the moduli set. The

approximate technique used a redundant residue to eliminate modulo-(M) operation, while the exact technique used a modified version of CRT. The modified CRT states that never exceed the dynamic range more than once (M).

The scaling error in the approximate technique was bounded.

Ulman published a modified version of the Szabo scaler in 1993 and since then, the results of all scalers have errors less than 1.5.

Scaling is one of the most important units and a necessary module to avoid overflow in a RNS-based system.

Various scalers for three moduli set ($2^n - 1$; 2^n; $2^n + 1$) have been developed and introduced in literature. Nowadays with the increasing demand for large dynamic range and parallel computing, four-moduli set are more attractive. For the first time, design and implementation of a four-moduli set RNS scaler is presented (Ulman and Zurada, 1993). In order to retain high performance while decreasing hardware complexity, two main points were considered by the authors:

- First, the four moduli set ($2^n - 1$; $2^n + 1$; 2^{2n}; $2^{2n} + 1$) is selected due to its large dynamic range and well-formed moduli set.
- Second, the Chinese Remainder Theorem with arithmetic simplifications is used to reduce complexity. The proposed scaler is synthesized using Synopsys Design Compiler (DC) and the Library for n = (4; 8; 16; 32; 64; 128) for all components and for separate modular channels.

Scaler's input and output are both residues considering the moduli set, and a scaler has modular channels as many as number of moduli set. Designing complex operations are usually performed using algorithms that are used in reverse conversion such as Chinese

Remainder Theorem (CRT), conversion in co-based algorithm, and Chinese Remainder **Theorems 1 and 2.**

One of the common methods to decrease the high power consumption of ROM matrices in ROM-based scaling schemes is to replace them by modular multiplexers. This method, however, increases the cost of implementation drastically, which would be even worse for large moduli sets. Hardware cost of all ROM based scalars increase by increasing the number of moduli. Hence, the authors can be manipulated with full adders for less area requirement (Ulman and Zurada, 1993)

In recent years, RNS became very popular in extended use of special-purpose processors, and increase in hardware capability of complex operations.

Optimizing performance of complex operations plays a significant role in the RNS-based system performance. A new efficient and low-cost scaler for a 6n bits dynamic range for four moduli set

(2^n -- 1; 2^n + 1; 2^{2n}; 2^{2n} + 1), which is under review was proposed and presented by (Ahangaran et al., 2014), which is designed based on the CRT algorithm, and is simplified to an efficient VLSI architecture. Proposing a scaler for larger dynamic ranges will ultimately overcome restrictions of using RNS in applications with large dynamic range requirements. However until that is than the non-iterative algorithm will achieved efficient time relative to the iterative algorithms.

2.7 Parity Checking

Before describing parity-checking technique, we define the interval $\left[0, \frac{M}{2}\right]$ to be positive numbers and the interval $\left[\frac{M}{2}, M\right]$*to* be negative numbers in the RNS, besides; we shall also

say that two numbers have the same parity if the authors are both even or both odd. Now, we describe parity-checking technique in the following.

Assume that the moduli $m_1, m_2 \ldots \ldots m_n$ are all odd numbers, and $X = (x_1, x_2 \ldots\ldots x_n)$ and $Y = (y_1, y_2 \ldots \ldots y_n)$. There are two numbers in the RNS, we have the following theorems.

Theorem 1

Let X and Y have the same parities, $Z = X - Y$ then, we have $X \geq Y$. If and only if Z is an even number. We also have $X < Y$ if and only if Z is an odd number.

Theorem 2

Let X and Y have different parities and $Z = X - Y$. Then, we have $X \geq Y$ if and only if Z is an odd number. We also have $X < Y$ if and only if Z is an even number. Let the value of X be the largest positive number. We can check the sign of Y by applying Theorems 1 and 2. Now, an example from (Yang *et al.*, 2004) is given below.

Assume that the moduli of the RNS are $m_1 = 3, m_2 = 5, m_3 = 7$ We get $M = 105$ .The positive and negative numbers fall into the interval [0, 52] and [53, 104], respectively.

Solution

Let $X = (1, 2, 3) = 52$ and $Y = (0, 4, 5) = |-51|_{105} = |54|_{105}$. We then say X and Y have the same parities. We compute $Z = X - Y = (1, 3, 5) = |103|_{105}$. Then, we have $< Y$ from Theorem 1. To check the number whether it is odd or even, the parity checking stores all the residues of those numbers modulo 2 by a Table. We know that a number is even or odd if its residues modulo 2 equal 0 or 1. Thus, we can easily get the parity of the number by looking up the Table. Furthermore, using Theorems 1 and 2, the parity checking technique makes the number comparison and the sign detection in RNS simple and efficient (Yang et al., 2004).

2.8 Review of Some Division Algorithm Dependent on the Parity Checking Technique

These algorithms include: Hiasat and Zohdy (1997), Yang et al., (2004) and Chang and Yang (2013).

2.8.1 HIASAT AND ABDEL-ATY-ZOHDY'S DIVISION ALGORITHM IN RNS (1997)

Hiasat and Abdel-Aty-Zohdy's (1997) were the first to use the parity checking technique to improve on division algorithms in RNS without using the CRT/MRC. In using parity checking technique, we can speed-up sign detection in RNS. However their algorithm cannot take negative dividend and divisor and the temporal quotient in their algorithm is smaller. This algorithm only uses addition, subtraction, and multiplication operations to solve division problem in the RNS. Furthermore, the algorithm performs the RNS division efficiently because it avoids the sign determination and the overflow detection. The following is an outline of their algorithm.

Step 1: *Set* the quotient $Q = 0$.

Step 2: Compute $j = h(X)$ and $k = h(Y)$, where j and k are the highest power of 2 in X and Y, respectively.

Step 3: If $j > k$, then we compute $Q = Q + 2^{j-k-1}$, $X = X - 2^{j-k-1} * Y$, $Q = Q$, $X = X$, and return to Step 2.

Step 4: If $j = k$, then compute $X\_ = X - Y$ and $j\_ = h(X\_)$. If $j\_ < j$ then $Q = Q + 1$.

Otherwise, Q is unaltered. End procedure.

Step 5: If $j < k$, then Q is unaltered. End procedure.

2.8.2 THE DIVISION ALGORITHM IN RNS BY YANG ET AL., (2004) USING THE PARITY CHECKING TECHNIQUE

Yang et al., (2004) proposed an algorithm using the parity checking technique to improve upon Hiasat and Abdel-Aty-Zohdy's (1997) division algorithm. Their algorithm, can speed-up sign detection in RNS and the dividend and divisor can be negative and the temporal quotient is also larger than that in Hiasat and Abdel-Aty-Zohdy's division algorithm. Approximating the actual quotient by adding the temporal quotients in their algorithm is two times faster than that in Hiasat and Abdel-Aty-Zohdy's algorithm. According to their simulation results, the authors concluded that their algorithm indeed can reduce the number of execution rounds by 50% relative to Hiasat and Zohdy. The authors assume that, if we want to compute $Q = \left[\frac{X}{Y}\right]$ in the RNS, where X, Y and Q are integers. h(I) is defined to be the highest power of 2 in the variable (I), where (I) is an integer. The authors illustrated their algorithm with an example. The steps of their algorithm are as follows;

Example,

We have $h(|10|) = 3$ and $h(|-7|) = 2$. Besides, we define S(X)to be the sign of variable (X). If S(X)= S(Y), it denotes that X and Y are with the same sign. The algorithm is describe as follows

Steps

1. Set the quotient $Q = 0$ and $C = 0$
2. Compute $j = h(X)$ and $k = h(Y)$ where j and k are the highest power of 2 in X and Y respectively. Then, check the signs of X and Y by parity checking. According to the relationships between j and k, we perform one of the following three steps.
3. If $j > k$ we perform the following operations. Set $C = 1$ if $S(X) = S(Y)$ Then we compute $\acute{Q} = Q + 2^{j-k}, \acute{X} = X - 2^{j-k} * Y, Q = \acute{Q}, X = \acute{X}$ otherwise we compute $\acute{Q} - \acute{2}^{j-k,}X\acute{} = X + 2^{j-k} * Y, Q = \acute{Q}$ and $X = \acute{X}$ Go step 2

4. If j = k, we perform the following operations. If $S(X) = S(Y)$ and $C = 1$ then we compute $\acute{X} = \acute{X} - Y$ and $j = h(\acute{X})$. If $\acute{j} < j$ then we compute Q = Q + 1. Otherwise, Q is unchanged. End procedure

   If $S(X) \neq S(Y)$ and c = 1, then we compute $\acute{X} = X + Y$ and check the sign of $\acute{X}$ by parity checking. If $\acute{X} > 0$ then we compute $Q = Q - 1$. Otherwise, we

   Set $Q = Q - 2$. End procedure. If $S(X) \neq S(Y)$ and $C = 0$, then we set $Q = -1$. End procedure.

5. If j< k, we perform the following operations. If c= 0, then Q is unchanged. End procedure.

2.8.3 CHANG AND YANG DIVISION ALGORITHM (2013)

After observing Yang et al. (2004) algorithm, Chang and Yang found out that $h(I)$ and the parity checking technique is performed twice in each round of that algorithm. So the authors realized that if the authors decrease the numbers of these two computations in each round, the execution time can further be reduced. So, Chang and Yang then proposed a bisection method in RNS to accomplish this purpose. Their algorithm steps are shown below:

Steps;

Inputs X, Y are expressed in RNS

Output, Y are expressed in RNS

1. Compute $j = h(X)$and $k = h(Y)$.Where (i and k) are the highest powers of two in X and Y respectfully.
2. If $j = k$, then use the parity checking technique to compare X and Y. if $X > Y$ then set $Q = 1$. Otherwise set $Q = O$.End program. If $j < k$, then set $Q = 0$. End the program.

3. Compute $Q_U = 2^{j-k+1}$ and $Q_L = 2^{j-k-1}$
4. Compute $\acute{Q} = Q_U + RNSQ_L$. If $\acute{Q}$ is odd, then set $\acute{Q} = \acute{Q} + RNS^{(1,1,\ldots.1)}$.
5. Compute $Q = \frac{\acute{Q}}{2}$ and $Z = X - RNS^Q \times RNS^Y$. Then use the parity checking technique to determine the sign and magnitude of Z.
6. If $Z > Y > 0$, then set $Q_L = Q$. Go to step 3.
7. If $Z < 0$, then set $Q_U = Q$.Go to step 3.
8. If $Y > Z > 0$, then Q is the quotient. End the program.

Example 2.5

The actual quotient $Q = \left[\frac{258}{33}\right] = 7$ in the binary form 7 is represented as

$(111)_2 = 2^2 + 2^1 + 2^0$. And note that Q was in the interval $Q_U = 2^{j+1-k}$ and $Q_L = 2^{j-k-1}$.

Illustration

Take the moduli set to be {7,9,11} calculating $Q = \left[\frac{X}{Y}\right]$ where $X = 258 \xrightarrow{RNS} (6,6,5)$ and $Y = 33 \xrightarrow{RNS} (5,6,0)$computing $j = h(X) = 8$ and $k = h(Y) = 5$. $Q_U = 2^{j-k+1}$and$Q_L = 2^{j-k-1} = 2^{8-5+1} = 2^4 = 16$ and$Q_L = 2^{8-5-1} = 2^2 = 4$

Hence it is true that Q lies between 16 and 4 which is 7 therefore $Q = \left[\frac{258}{33}\right] = 7$.

2.9 Review of Some Algorithms of RNS Implementation of RSA

This section reviewed literature on some RNS division algorithm that has been applied to RSA. Specifically Imbert and Bajard (2002) and Hung and Parhami (1994).

2.9.1 RSA IMPLEMENTATIONS BY LAURENT IMBERT AND JEAN-CLAUDE BAJARD 2002

Laurent Imbert and Jean-Claude Bajard (2002) proposed an efficient hardware implementation of RSA based on the Residue Number System (RNS), which allows for fast parallel arithmetic. The authors proposed RNS versions of Montgomery multiplication and exponentiation algorithms and the authors illustrated the efficiency of their approach with two implementations of RSA. The authors compared their work to previously proposed methods and their solution requires less elementary operations and is very promising. The authors presented their work in two ways.

- RSA without conversions
- RSA with conversions.

The authors concluded their work by the analyses that the first solution uses an embedded RNS arithmetic, which maps the values in RNS before the computations and convert them back in binary at the end. But the real novelty is a full RNS cryptosystem. The message is never considered as a binary number but rather in RNS all along the protocol. Thus no conversion is needed. This approach requires both parties to agree on a set of RNS parameters beforehand. The authors compared their work to previously proposed algorithms and their algorithm requires less elementary operations and uses only integer arithmetic

Besides the conditions on their parameters are easier to satisfy than the ones the authors compared with in other methods. However, their algorithm is still having long delay time once it has an overall loop because the type of division method the authors adopted is the MM.

2.10 Chin Yu Hug and Behrooz Parhami (1994) Fast RNS Application to RSA Cryptography

Chin and Behrooz in 1994 considered the problem of division by fixed divisors in RNS. Ordinary integer division is performed; i.e., given the dividend X and the divisor D, we wish to find the quotient Q = (X/D) and the remainder (R = X – QD). Their idea is to perform some operations based on the divisor to improve the on-line speed of divisions. According to them several algorithms for general residue division have been proposed in the past under the assumption that D is fixed and X is uniformly distributed, the fastest of them has a worst-case time complexity of O(log, X) = O(nb), where n is the number of moduli and b is the number of bits in the largest modulus. In their paper the authors presented two division algorithms for fixed divisors.

Encryption and decryption in RSA cryptography are modular exponentiation operations of the form Z = XY mod D. For encryption, X is the plain text, Y and D together comprise the encryption key, and Z is the ciphered text. For decryption, X is the ciphered text, Y and D the decryption key, and Z is the deciphered text. All operands, X, Y, D are potentially very large integers, perhaps 1000 bits long.

The authors compared their algorithm with existing ones in terms of computation time for each modulo D multiplication fixed-divisor algorithms is apply to the modulo D reduction step that follows a regular multiplication. The dynamic range of RNS thus needs to be at least the square of the modulus (D). We almost always have to compare actual or estimated encryption rates of the designs. The authors compared their proposed method with two classical sequential methods: one uses a binary version of multiplying by divisor reciprocal for modular reduction and the other uses a residue table for modular reduction. With a b-bit processor, a modular multiplication takes 9(m/b) 2 and 4(m/b)* steps, respectively. However

the problem with their algorithm is the division method the authors adopted. The division involved is iterative and does not support all integers as divisors within the dynamic range.

2.11The Mathematics of the RSA Public-Key Cryptosystem

RSA public-key cryptosystem was invented at MIT in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman. The public key in this cryptosystem consists of the value n, which is called the modulus, and the value e, which is called the public exponent. The private key consists of the modulus n and the value d, which is called the private exponent.

An RSA public-key / private-key pair can be generated by the following steps:

1. Generate a pair of large, random prime's p and q.

2. Compute the modulus n as n = p.q.

3. Select an odd public exponent e between 3 and n-1 that is relatively prime to p-1 and q-1.

4. Compute the private exponent d from e, p and q.

5. Output (n, e) as the public key and (n, d) as the private key.

The encryption operation in the RSA cryptosystem is exponentiation to the eth power modulo n:

$c = \text{ENCRYPT}(m) = m^e \bmod n$.

The input m is the message; the output c is the resulting cipher text. In practice, the message m is typically some kind of appropriately formatted key to be shared. The actual message is encrypted with the shared key using a traditional encryption algorithm. This construction makes it possible to encrypt a message of any length with only one exponentiation.

The decryption operation is exponentiation to the d^{th} power modulo n:

$m = DECRYPT(c) = c^d \bmod (n)$

The relationship between the exponents' e and d ensures that encryption and decryption are inverses, so that the decryption operation recovers the original message m. Without the private key (n, d) (or equivalently the prime factors p and q), it is difficult (by conjecture) to recover m from c. Consequently, n and e can be made public without compromising security, which is the basic requirement for a public-key cryptosystem.

The fact that the encryption and decryption operations are inverses and operate on the same set of inputs also means that the operations can be employed in reverse order to obtain a digital signature scheme following Diffie and Hellman's (1976) model. A message can be digitally signed by applying the decryption operation to it, i.e., by the exponent to the d^{th} power:

$s = SIGN(m) = m^d \bmod (n)$.

The digital signature can then be verified by applying the encryption operation to it and comparing the result with and/or recovering the message:

$m = VERIFY(s) = s^e \bmod n$

In practice, the plaintext m is generally some function of the message, for instance a formatted one-way hash of the message. This makes it possible to sign a message of any length with only one exponentiation.

Table 2.2 The Encryption And Decryptions of Values of M From 0 To 9 and the Resulting Cipher Texts.

Key Pair Public key: n = 55, e = 3 Private key: n = 55, d = 7					Key Pair Generation Primes: p = 5, q = 11 Modulus: n = p.q = 55 Public exponent: e = 3 Private exponent: $d = 3^{-1} \bmod 20 = 7$	
Message		Encryption $c = m^3 \bmod n$			Decryption $m = c^7 \bmod n$	
m	$m^2 \bmod n$	$m^3 \bmod n$	$c^2 \bmod n$	$c^3 \bmod n$	$c^6 \bmod n$	$c^7 \bmod n$
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	4	8	9	17	14	2
3	9	27	14	48	49	3
4	16	9	26	14	31	4
5	25	15	5	20	15	5
6	36	51	16	46	26	6
7	49	13	4	52	9	7

Source; Burt Kaliski RSA Laboratories

From the table 2.2, encryption of values m from 0 to 9 as well as decryptions of the resulting cipher texts, the exponentiation is optimized. To compute m^3 mod (n), one first computes m^2 mod n with one modular squaring, then m^3 mod n with a modular multiplication by m. The decryption is done similarly: One first computes c^2 mod n, then c^3 mod n, c^6 mod n, and c^7 mod n by alternating modular squaring and modular multiplication.

CHAPTER THREE

METHODOLOGY

3.1 The Algorithm

We proposed an algorithm based on Mansoureh and Mohammed (2012) Pure RNS Division Algorithm. Two different moduli sets were considered that is$\{2^n - 2, 2^n - 1, 2^n + 1\}$ for n = 2 (Mansoureh and Mohammed, 2012), we had (2, 3, 5) and for the second moduli-set$\{ 2^n - 1, 2^n + 1, 2^n + 3\}$ for n = 3 (Yang et al., 2004) we had (7, 9, 11). The dynamic ranges are 29 and 692 respectively. We then did our analysis on integer quotient and non-integer quotient for the range of value for within 29 and also for 692. In both cases we run several examples for non-integer and integer quotient regarding improper fraction. Finally, we compared our algorithm to other state of the art algorithms to draw conclusions.

3.2 The Pure RNS Division Algorithm by Mansoureh and Mohammad (2012)

Input: $X = (x_1, x_2 \ldots\ldots\ldots x_L)$, $Y = (y_1, y_2 \ldots\ldots\ldots y_L)$

Input $m_1, m_2 \ldots\ldots m_L$ the moduli set

Output: K=X/Y and R=(X mod Y) in RNS, Condition; all moduli $m_1, m_2 \ldots\ldots m_L$ are relatively prime numbers but 2.

Steps

1) Calculate Y^{-1}= multiplicative inverse of Y
2) Calculate R = X mod Y
3) Calculate K= (X – R). Y^{-1}
4) If any, $y_i = 0$, then set $k_i = k_n mod m_i \; in \; which k_n \neq 0$.
5) End program.

The pure RNS division algorithm by Mansoureh and Mohammad (2012) is similar to Chang and Yang (2013) algorithm in terms of generality. Again it's also a non-iterative algorithm thus; it is faster than all other division though it also uses the CRT/MRC. It has drastically reduced time and as such is a good algorithm that can be applied to the RSA cryptosystem without restrictions.

However there are other division algorithms which did not depend on the traditional CRT/MRC. These methods use the parity checking technique, the highest powers of two's and the bisection method to perform division in RNS.

3.3 The RSA Cryptosystem

This cryptosystem uses computations in $\mathbb{Z}_n$ where n is the product of two distinct odd primes p and q. for such an integer n, note that

$$\emptyset_{(n)} = (p-1)(q-1) \qquad (3.1)$$

Let $n = pq$ where p and q are primes. Let $\mathcal{P} = \mathbb{Z}_n$ and define

$$\mathcal{K} = \{(n, p, q, a, b): ab \equiv 1(mod)\phi_{(n)}\} \qquad (3.2)$$

For$\mathcal{K} = (n, p, q, a \; and \; b)$

$$e_k(x) = x^b mod(n) \qquad (3.3)$$

$$d_k(y) = y^a mod(n) \qquad (3.4)$$

$x, y \in \mathbb{Z}_n$

The values n and b compose the public key, and the values (p, q and a) form the private key.

3.4 Construction of the Algorithm

From Equation (3.3)

We let $M = \left(\frac{X}{Y}\right)$ where M is the quotient, X is the numerator and Y is the denominator.

Thus modified as:

$$e_k\left[\frac{X}{Y}\right] = \left[\frac{X}{Y}\right]^b mod(n) \qquad (3.5)$$

From Equation (3.4)

Let $Q = \left(\frac{D}{C}\right)$ where Q is the cipher text, D is the dividend and C is the divisor

Thus becomes

$$d_k\left(\frac{D}{C}\right) = \left[\frac{D}{C}\right]^a mod(n) \qquad (3.6)$$

This algorithm maintained the number of keys, both public keys and private keys.

Thus: (n, b) composed the public keys and (p, q, a) forms the secret keys. By considering the Moduli-sets; $\{2^n - 2, 2^n - 1, 2^n + 1\}$, for n=2; (2, 3, 5)and $\{2^n - 1, 2^n + 1, 2^n + 3\}$, for n=3; (7, 9, 11), we use the following assumptions;

Assumptions

The following are assumptions to our proposed algorithm

- $X > Y > 0$
- No denominators or divisors should be a multiple of the chosen moduli set. Else the algorithm will break down due to a zero multiplicative inverse in one of the modulus
- After going through the division phase of the algorithm, when the CRT display the decimal equivalent of the RNS say (k) and its greater than (n),then the expected plain text would be (k)n. that is $(k)n.^b\, mod(n) = (k)^b mod(n)$

The Encryption

$e_k(M) = M^b mod(n)$ In the usual RSA encryption

Let $M = Q = \left(\frac{X}{Y}\right)$ where Q is the quotient, X is the dividend and Y is the divisor

Steps

1) Input $\{m_1, m_2 m_3\}$ the moduli set,
2) Input X and Y express in RNS
3) Input (n and b) the public keys
4) $e_k\left[\frac{X}{Y}\right] = \left[\frac{X}{Y}\right]^b mod(n)$
5) Calculate Y^{-1}= multiplicative inverse of Y
6) Calculate R = X mod Y
7) Calculate Q= (X – R). Y^{-1}
8) If any, $y_i = 0$, then set $Q_i = Q_n mod m_i,\ Q_n \neq 0$.
9) Using CRT convert $(j,k,l)_{RNS}$ to Q i.e. the decimal value
10) Hence $e_k(Q) = Q^b mod(n) = G$
11) ELSE $e_k(j,k,l) = (j,k,l)^b mod(n)$
12) End the program.

Note the Transformation $\left(\frac{X}{Y}\right) = (j,k,l)_{RNS} = Q$. Is based on Mansoureh and Mohammed (2012), and Q output will appears in RNS representation.

THE DECRYPTION

$d_k(G); G^a \ mod(n) = Q = M \ = \frac{X}{Y}$ The cipher text in the usual RSA decryption

Steps

1) Input $\{m_1, m_2 m_3\}$ the moduli set,
2) Input (t, f, h)$_{RNS}$
3) Input (p, q and a) the secret keys
4) Using the CRT convert (t, f, h) to G the decimal value.
5) $d_k[G]^a mod(n) = Q$. ELSE
6) $d_k[t, f, h]^a = \ (j, k, l) = M = \frac{X}{Y}$

   End the program.

Table 3.1: Sample of examples for the moduli set (2, 3, 5) for values of X and Y ranging between (0-29)

S/N	Integer Quotient	Non-Integer Quotient	Proper Fraction
1	21/7	3/2	5/10
2	25/19	6/4	3/9
3	28/7	29/11	7/28
4	24/13	29/28	2/24
5	22/11	18/17	17/18
6	20/23	25/23	8/17
7	25/5	27/8	12/14
8	24/8	13/4	19/20
9	18/7	16/5	28/29
10	12/17	9/7	15/16

Table 3.2: Sample of examples for the moduli set (7, 9, 11) for values of X and Y ranging between (0-692)

S/N	Integer Quotient	Non-Integer Quotient	Proper Fraction
1	692/2	691/3	9/316
2	595/5	542/16	15/366
3	550/11	597/12	28/448
4	572/13	578/17	99/129
5	352/16	253/18	299/389
6	616/8	316/9	16/352
7	354/6	366/15	11/550
8	258/33	448/28	13/570
9	110/22	129/99	5/595
10	387/43	389/299	2/692

Illustrative Example

The following example illustrates our algorithm.$\left[\frac{258}{17}\right]$

Keys Selection

This cryptosystem uses computations in $\mathbb{Z}_n$.Let $n = p.q$ where n is the product of two distinct odd primes p and q. for this purposes let $p = 11\ and\ q = 17$ implies $n = 187$ calculating $\emptyset_{(n)} = (11-1)(17-1) = 160$. let our moduli set be{7,9,11}

$\mathcal{K} = \{(n, p, q, a, b): ab \equiv 1(mod)\phi_{(n)}\}$ Calculate $ab \equiv 1(mod)\phi_{(n)}$ let $b = 7$ using the extended Euclidean algorithm$7 \times 7^{-1} \equiv 1\ mod(160)$ then $a = 23$.

For $\mathcal{K} = (n, p, q, a, and\ b) = (187,11,17,7, and\ 23)$ respectively

Example

$\left[\frac{578}{17}\right]$fixed divisor and an integer quotient.

The Encryption Algorithm

Let $X = 578,\ Y = 17$

Steps

1) Input $\{7,9,11\}$ the moduli set
2) Input $X = 578 \xrightarrow{RNS} (4,2,6)$and $Y = 17 \xrightarrow{RNS} (3,8,6)$
3) Input (n and b) the public keys $n = 187, b = 7$
4) $e_k\left[\frac{578}{17}\right] = \left[\frac{578}{17}\right]^7 mod(187)$
5) Calculate Y^{-1}= multiplicative inverse of Y Else If any, $y_i = 0$, then set $Q_i = Q_n mod m_i$ in which$Q_n \neq 0$ i.e 17^{-1} w.r.t (7, 9, 11)=(5, 8, 2)
6) Calculate R = X mod $_Y$=$(578)_{17}$=0 $\xrightarrow{RNS}$ (0,0,0)
7) Calculate Q= (X – R). Y^{-1}=$[(4,2,6) - (0,0,0)].[(5,8,2)] = (6,7,1)$
8) Using CRT convert $(6,7,1)_{RNS}$ to (34)i.e. the decimal value
9) Hence $e_k(34) = 34^7 mod(187) = 34$
10) ELSE $e_k(6,7,1) = (6,7,1)^{17} mod(187)$
11) Hence $34 \xrightarrow{RNS} (6,7,1,)$ is send as the encrypted massage where X and Y or Q is the plain text.
12) End the program.

The Decryption Algorithm

Steps

Let $G = \left(\frac{D}{C}\right)$ where G is the cipher text, D is the dividend and C is the divisor.

1) Input $\{7,9,11\}$ the moduli set,
2) Input $(G)_{RNS} = (6,7,1)$
3) Using the CRT convert $(G)_{RNS} = (6,7,1)$ to decimal 34
4) Input $(p, q \text{ and } a) = 23$ the secret key
5) $d_k[6,7,1]^{23} mod(187) = d_k[34]^{23} mod(187) = (6,7,1)$
6) Using (CRT)transform $(6,7,1)_{RNS}$ to 34
7) Hence $Q = 34$ is decrypted
8) End the program.

CHAPTER FOUR

DISCUSSION AND ANALYSIS OF RESULTS

4.1 Performance Analysis of the Proposed Algorithm

The proposed algorithms discussed in chapter three were simulated and analyzed using **C++** and results are summarized below.

4.2 Security Analysis

Security analysis was performed to test the effectiveness of the proposed algorithm on RSA attacks (key space, and key sensitivity analysis). Experimental results showed that the proposed algorithm is highly secured against such attacks.

4.2.1 KEY SPACE ANALYSIS

The brute force attack is computationally infeasible for cryptosystems with sufficiently large key space. The proposed algorithm is a public key cryptosystem which has

$\mathcal{K} = (n, p, q, a, and\ b)$. The proposed algorithm achieves an efficient coding process when the key space K is large, note; (n) depends on (p) and (q). However, we adopt (1024) bits as our key space per the Digital encryption standards (DES). This gives us the combination of choices. $k = 2^n$.

$2^{1024} = 1.797693134862315907729305190789e + 308.$

Diffie and Hellman (1976) outlined a "brute force" attack on DES. Brute force attack means trying as many as 2^{56} possible keys before you can decrypt the cipher-text into a sensible plaintext message. The authors proposed a special purpose "parallel computer using one million chips to try one million keys each" per second. Our key space is $2^{1024} > 2^{56}$.

It is obvious here that our key space is greater than that of the brute –force attack key space.

4.2.2 KEY SENSITIVITY ANALYSIS

The greatest sensitivity analysis in our algorithm is explained in example 2.3 (a, and b) respectively. In (i) division in RNS is equivalent to division in conventional representation. Where as in part (ii), we notice that division in RNS is not equivalent to that in conventional representation when the quotient is a non-integer value. This can enhance the sensitivity of the key because $\frac{6}{2} = 3$ both in RNS and in conventional but $\frac{6}{4} = 5$ in RNS but $= 1.5$ in conventional representation. That is with respect to mod (7).

A good cryptosystem should be sensitive to secret keys. A slight Change in the key value should lead to a significant Change in either a plain text or a cipher text. The addition of the used of the moduli set also help in security of the algorithm. There must be moduli set to be agreed on by both parties. The parameters (p, q, n, a, and b) in our examples resulted in significant difference with the actual answers during our experiment. The parameters for $\mathcal{K} = (n, p, q, a, and\ b) = (35,5,7,17, and\ 17)$ respectively which we used in illustrating our example as shown in table 4.1. The choice of the moduli set alone could be a very strong security feature. This is because the choice may vary from party to party. Again the moduli set we used here has a dynamic range of M-1 representation that has to do with the repetition of RNS representation outside the dynamic range.

Besides, negatives numbers are acceptable. Example, the cipher text (0, 4, 5) could be $(-51)_{105}$ or $(54)_{105}$ and can still has another beyond the dynamic range of 105.

Table 4.1 Sensitivity Analysis of Integer and Non-Integer Quotient

Example	Type of quotient	Plaintext in conventional division	Plaintext in RNS	Moduli set	Cipher text
6/4	Non integer	1.5	$19 \xrightarrow{RNS} (5,1,8)$	(7,9,11)	$24 \xrightarrow{RNS} (3,6,2)$
6/2	Integer	3	$3 \xrightarrow{RNS} (3,3,3)$	(7,9,11)	$33 \xrightarrow{RNS} (5,6,0)$
258/33	Non integer	7.8181	$11 \xrightarrow{RNS} (4,2,0)$	(7,9,11)	$16 \xrightarrow{RNS} (2,7,5)$

4.3 Time Complexity Analysis

Algorithm efficiency is concerned with utilization of two important resources time and space, time complexity refers to the time taken by an algorithm to complete and produce a result. An improvement in time complexity increases the speed with which the algorithm proceeds to completion. Space complexity refers to the amount of space taken by an algorithm to produce a result. An improvement in space complexity reduces the amount of space (memory, disk space, etc.) taken by an algorithm (Steve and Stephen, 2004)

We analyzed the time taken for our proposed algorithm to produced results or an output. The results reveal that, encryption and decryption time's increases depending on the method of RNS division involved, whether iterative or non-iterative. We realized that when encrypting with a non-integer quotient encryption time increases as the divisor and dividend becomes larger, however larger non-integer quotient still has less executing time relative to smaller integer during encryption. However, it was observed that, lesser divisor and lesser dividend in non-integer quotient would rather have larger executing time compared to larger dividend and divisor. That of integer quotient will still be higher but lesser than smaller values in integer decryption.

4.4 Executing Time Analysis for the Moduli-set (2, 3, 5)

The freedom of number representation in RNS is limited by the Dynamic range, which is dependent on the moduli set. It is observed that when the moduli set is chosen such that it has a small dynamic range, the algorithm would be limited to just a few numbers qualified for encryption and thus, attackers can easily break the algorithm. The encryption time for these moduli set is higher than the decryption time.

The average time to do a complete encryption and decryption with respect to an integer quotient is 16822.3micro seconds, which is approximately 17seconds, and that for a non-integer quotient is 15228.1micro seconds approximately 15seconds. Tables 4.2 and 4.3 illustrate these.

Table 4.2: Time Analysis for Integer Quotient in Microseconds. Moduli set (2, 3 5)

S/N	Example	Encryption Time	Decryption Time	Average Time
1	21/7	16642	22865	19753.5
2	14/7	21011	16128	18569.5
3	28/7	14852	13905	14378.5
4	26/13	21308	13725	17516.5
5	22/11	13098	14689	13893.5
Total Average				16822.3

Table 4.3: Time Analysis for Non-Integer Quotient. Moduli set (2, 3 5)

S/N	Example	Encryption Time	Decryption Time	Total Time
1	25/13	19102	15110	17106
2	29/11	16724	13315	15019.5
3	29/17	15847	15221	15534
4	25/7	13659	15675	14667
5	19/7	14313	13315	13814
Total Average				15228.1

4.5 Executing Time Analysis for the Moduli-set (7, 9, 11)

These moduli-set used larger dynamic range, which gives room for larger domain for values of X and Y. the average time for integer quotient in this scheme is16822.3microseconds approximately 17seconds and that of non-integer is 15228.1microseconds approximately 15sec. as shown in Tables 4.4 and 4.5 respectively. Table 4.6 did the comparison between the average times for the two different moduli sets, and it was observed that The authors both have almost the same average time, we compared our algorithm to some state of the art algorithms, and it revealed that our algorithm has the best average time for both integer and non-integer quotient with respect to the two different moduli sets used, as shown in Table 4.11 showed this whiles Tables 4.7, 4.8, 4.9 and 4.10 showed the average time for the state of the art algorithm with respect to the two different moduli set.

Table 4.4: Time Analysis for Integer Quotient. Moduli set (7, 9, 11)

S/N	Integer Quotient	Encryption Time	Decryption Time	Average Time
1	692/2	22900	20240	21570
2	595/5	16817	17042	16929.5
3	572/13	18894	16409	17651.5
4	352/16	19641	19334	19487.5
5	616/8	17028	21025	19026.5
Total Average				18933

Table 4.5: Time Analysis for Non-Integer Quotient. Moduli set (7, 9, 11)

S/N	Non-Integer Quotient	Encryption Time	Decryption Time	Average Time
1	389/299	19773	22615	21194
2	316/15	13442	20854	17148
3	316/19	18430	20508	19469
4	448/38	13682	29409	21545.5
5	129/100	22902	13391	18146.5
Total Average				19500.6

Table 4.6: Average Time Complexity for moduli-sets (2, 3, 5) and (7, 9, 11) in microseconds

Type of quotient	Average time (7, 9, 11)	Average time (2, 3, 5)
Non integer	19500.6	15228.1
Integer	18933.0	16822.3

Table 4.7 Average Time Complexity for Chin, Y. H. and Behrooz, P. (1994) for the Moduli Set (2,3, 5) Integer Quotient

S/ N	Example	Encryption Time	Decryption Time	Average Time
1	21/7	21311	19926	20618.5
2	14/7	19265	18453	18859.0
3	28/7	18564	21146	19855.0
4	26/13	22143	20215	21179.0
5	22/11	17231	16432	16831.5
Total Average				19468.6

Table 4.8: Average Time Complexity for Chin, Y. H. and Behrooz, P. (1994) for the Moduli Set (2, 3, 5) Non-Integer Quotient

S/N	Example	Encryption Time	Decryption Time	Average Time
1	25/13	23451	19867	21659.0
2	29/11	24732	21020	22876.0
3	19/7	22314	19981	21147.5
4	26/17	19556	18923	19239.5
5	23/11	18312	20013	19162.5
Total Average				20816.9

Table 4.9: Average Time Complexity for Chin, Y. H. and Behrooz, P. (1994) for the Moduli Set (7,9, 11) Non-Integer Quotient

S/ N	Non-Integer Quotient	Encryption Time	Decryption Time	Average Time
1	389/2	28359	22615	25487.0
2	316/3	26527	21853	24190.0
3	316/6	21563	20532	21047.5
4	448/5	28479	24091	26285.0
5	129/4	22711	23351	23031.0
Total Average				24008.1

Table 4.10: Average Time Complexity for Chin, Y. H. and Behrooz, P. (1994) for the Moduli Set (7, 9, 11) Integer Quotient

S/N	Integer Quotient	Encryption Time	Decryption Time	Average Time
1	692/2	23659	21242	22450.5
2	595/5	26689	22411	24550.0
3	572/4	25984	23409	24696.5
4	684/6	26297	20963	23627.5
5	616/8	25795	21095	23445.0
Total Average				23753.9

Table 4.11: Comparison of our average times to some state of the art Algorithm in microseconds

Type of Algorithm	Average Time With Respect to Moduli Set	
	(2,3,5)	(7,9,11)
Our Algorithm		
Non-Integer Quotient	15228.1	19500.6
Integer Quotient	16822.3	18933.0
Chin, Y. H. and Behrooz, P. (1994)	(2,3,5)	(7,9,11)
Non-Integer Quotient	20816.9	24008.1
Integer Quotient	19468.6	23753.9

4.6 Error Analysis of Our Proposed Algorithm

We did the error analysis on our proposed algorithm in terms of the average executing time. This was performed on the different moduli set for (2, 3, 5) and (7, 9, 11). Experimental results revealed that, the mean, standard deviation and standard error are very minimal. Table 4.12 shows these statistics. We assume a significant level of 0.05.

The confidence interval of the two different moduli sets are shown in table 4.13.

In Table 4.14 we illustrated the percentage difference between our proposed algorithm and that of the state of the art we did the comparison with. The results showed that, our proposed algorithm is 7.29%, 15.51%,11.29% and10.36% better than that of the state of the art we did the comparison in terms of the moduli sets and the type of quotient involved.

Table 4.12: Error Analysis of the Proposed Algorithm

Descriptive Statistics

	N	Minimum	Maximum	Sum	Mean		Std. Deviation
	Statistic	Statistic	Statistic	Statistic	Statistic	Std. Error	Statistic
Time Encryption	10	2308	21011	147556	14755.60	1583.749	5008.255
Time Decryption For (2,3,5)	10	1395	22865	141438	14143.80	1662.706	5257.938
Time Encryption	10	13442	22902	183509	18350.90	1029.528	3255.655
Time Decryption for (7,9,11)	10	13391	29409	200827	20082.70	1346.445	4257.832
Valid N (Listwise)	10						

Table 4.13: Confidence Interval for the Error in Our proposed Algorithm

Moduli Set	Confidence Interval	
(2,3,5)	Lower bound	upper bound
Encryption	14677.60	14833.60
Decryption	14063.88	14223.72
(7,9,11)	Lower bound	upper bound
Encryption	18288.01	18413.79
Decryption	20010.78	20154.62

Table 4.14: Percentage Difference between the proposed Algorithm and Parhami

Quotient	Parhami		Proposed Algorithm		Percentage Difference	
	(2,3,5)	(7,9,11)	(2,3,5)	(7,9,11)	(2,3,5)	(7,9,11)
Delay for Integer	19468.6	23753.9	16822.3	18933.0	7.29%	11.29%
Delay For Non-Integer	20816.9	24008.1	15228.1	19500.6	15.51%	10.36%

CHAPTER FIVE

SUMMARY OF FINDING, CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

In this chapter, we shall summarize the major features of this work. This work is based on construction of a proposed algorithm for RNS implementation of RSA. RNS is used because of its benefits of improving parallelization.

5.2 Summary

Generally, this thesis presented an efficient RNS division algorithm for RSA implementation based on Mansoureh and Mohammed (2012) division algorithm. Two different moduli set were used, of which one has a larger dynamic range and the other a smaller. Excellent results were obtained in terms of disk space and memory management, time complexity, key space and sensitivity analysis. The proposed algorithm can absorb and control the emerging problem of growing key length in RSA and it is a general algorithm that can be applied to RSA cryptography in RNS such that fractions, decimals, integers can all be encrypted and decrypted. The proposed algorithm performed relatively well comparing it to existing state of the art algorithms

5.3 Conclusions

In this thesis it was identified that Performing division in RNS using the MRC and CRT was relatively time consuming compared to performing division in RNS using the parity checking technique and the highest powers of two. However some division algorithms are dependent on the CRT/MRC but non-iterative. The algorithm by Mansoureh and Mohammed (2012), which is non- iterative, is the bases of this algorithm, a comprehensive discussion on the proposed algorithm with illustrative examples were presented. We illustrated the RSA cryptosystem by showing how keys are generated, the encryption and decryption equation

were described and we translated them to the form of division encryption and decryption equations. The encryption processes were described through to the encryption algorithm. We constructed both the decryption and decryption algorithms and illustrated them with an example. Experimental results of our algorithm by comparing the type of quotient, the moduli sets, time complexity, key space and sensitivity of the propose algorithm indicates that;

When using the proposed algorithm;

- Choose a larger dynamic range for larger domain
- Non-integer quotient is highly secured and sensitive relative to integer quotient, and it is having the best time as well.
- It was observed that our algorithm is having the best average time for both integer and non-integer quotient relative to the state of the art algorithm that we compared ours to.
- The error analysis was performed and we had a negligible error margin to our algorithm.

5.4 Recommendations and Future Work

This thesis has made some gains. However, development can be looked at in the following areas

- The hardware implementation of the proposed system.
- Analyzing the performance of the proposed techniques and other division algorithm technique.
- The moduli set could also be enhanced to increase the dynamic range relative to the key space.
- The proposed algorithm can be tested on other public key cryptosystem like the Elgamal.

- With the idea of the proposed algorithm scaling in RNS could also be exploited for possible application to RSA

REFERENCES

Alhassan S. and Gbolagade, K.A. (2013). Enhancement of the security of a Digital Image Using the moduli set$\{2^n - 1, 2^n, 2^n\}$, International Journal of Advanced Research in Computer Engineering and Technology, ISSN; 2278-1323,PP.2223-2229,Vol.2,Issue 7.

Behrooz Parhami (1999). Computer Arithmetic Algorithm and Hardware Design. New York Oxford University Press

Burnett, S. and Paine S. (2004). RSA Security Officials McGraw-Hill

Chin, Y. H. and Behrooz, P. (1994). Fast RNS Division Algorithms for Fixed Divisors with Application to RSA Encryption, Department of Electrical and Computer Engineering University of California, USA. Information Processing Letters 51:163- 169

Chin-Chen, C. and Jen-Ho, Y. (2013). A Division Algorithm Using Bisection Method in Residue Number System International Journal of Computer, Consumer and Control.**2** (1): 59-66.

Daabo, M.I. and Gbolagade, K.A. (2012). Overflow Detection Scheme in RNS Multiplication before Forward Conversion, Journal of Computing.**4** (12):13-16.

Diffie and Hellman (1976). New Direction in Cryptography, IEEE Transactions on Information Theory archives vol. 22 Issues 6Th November.

Gbolagade, A. K. (2010). Effective Reverse Conversion in Residue Number System Processors. Delft, the Netherlands.

Gbolagade K. A. and Cotofana, S.D. (2009). A Reverse Converter for the new 4-moduliset$\{2n + 3, 2n + 2, 2n + 1, 2n\}$, Proceedings of the 2009 IEEE International Conference on Electronics Circuits and Systems(ICECS09), pp. 113-116, Hammamet, Tunisia

Gbolagade, K.A. and Cotofana, S.D. (2009).Residue-to-Decimal Converters for Moduli sets with Common Factors. IEEE.624-627

Griffin, M., Taylor, F. and Sousa, M. (1988). New scaling algorithms for the Chinese remainder theorem, in Signals, Systems and Computers, Twenty-Second Asilomar Conference on, vol. 1. IEEE, pp. 375–378

Griffin, M. S. M. and Taylor, F. (1989). Efficient scaling in the residue number system," in Int. Conf. Acoust., Speech, Signal Process, Glasgow, U.K., May, pp. 1075–1078.

Hiasat and Zohdy (1997). Design and Implementation of an RNS Division Algorithm.IEEE.240-249

Hitz, M. and Kaltofen M. (1995), Integer division in residue number systems IEEE Transactions on Computers.**44**(8):983– 989

Hussein, E., Hasan, M. A. and Elmasry M. I. (1998).A Low Power Algorithm for Division in Residue Number System IEEE

Jean-Claude, B. G. (1991). New Approach to Integer Division in Residue Number Systems, Proceedings of 10th IEEE symposium on Computer Arithmetic, Grenoble, France. 84–91

Jullien, G. A. (1978). Residue number scaling and other operations using rom arrays, Computers, IEEE Transactions on, vol. 100, no. 4, pp. 325–336,

Laurent, I. and Jean-Claude, B. (2002).A Full RNS Implementation of RSA IEEE Transactions on Computers.**53(**6):789-774.

Laurent, I. and Jean-Claude, B. (2004).A Full RNS Implementation of RSA Transactions on computers IEEE.**53**(5):1-6.

Lim, Z. and Philips, B.J. (2007).An RNS-Enhanced Microprocessor Implementation of public key cryptography in 41st Asilomar Conference on signals, system, and computers, Pacific Grove, California, USA.1430-1434.

Mansoureh and Mohammed (2012). Non Iterative RNS Division Algorithm IMECSVolI ISSN; 2078-0966 online

Mi, L. (2004).Arithmetic and Logic in Computer Systems, New Jersey, John Wiley And Sons, Inc., Hoboken

Miller, D. D. and Polky, J. N. (1984). An implementation of the LMS algorithm in the residue number system, Circuits and Systems, IEEE Transactions on, vol. 31, no. 5, pp. 452–461

Nobuhiro, T. and Teruki, I. (2002).Design of High-Speed RSA Encryption Processor Based on the Residue Table for Redundant Binary Numbers Systems and Computers in Japan.**33**(5):423-432

O'Keefe, K. H. and Wright, J. L. (1973). Remarks on base extension for modular arithmetic, Computers, IEEE Transactions on, vol. 100, no. 9, pp. 833–835,

Omar, A. F. (2011). Data Conversion in Residue Number System Department of Electrical & Computer Engineering McGill University Montreal,

Omondi A. and Premkumar, B., (2007).Residue Number System Theory and Implementation. Imperial College Press

Pemmaraj V.A.M (2009). RNS-To-Binary Converter for a New Three-Moduli set. IEEE Transactions on Circuits and Systems-II; Express Briefs, Vol. 54 No. 9.

Rivest, R., Shamir, A. and Adleman L. (1978). A Method for Obtaining Digital Signatures and Public Key Cryptosystems Communications of the ACM.21 (2):120–126

Shahram, M. and Davar K.T (2012). The Application of The Residue Number System in Digital Image Processing; Propose a Scheme of Filtering in Spatial Domain. Research Journal of Applied Sciences 7(6), PP. 286-292.

Shenoy, M. A. P. and Kumaresan, R. (1989). A fast and accurate RNS scaling technique for high speed signal processing, IEEE Trans. Acoust, Speech, Signal Process, vol. 37, no. 6, pp. 929–937.

Steven Burnett and Stephen Paine (2004). RSA Security Official Guide to Cryptography, Keller Graduate School of Management of DeVry University Edition, ISBN 0-07-225494-7.

Szabo, N. and Tanaka, R. (1967). Residue Arithmetic and Its Applications to Computer Technology, New York, McGraw Hill

Talesshmekael, D. K. and Mousave, A. (2012). Using One Hot Residue (OHR) in Image Processing; Proposed a Scheme of Filtering in Spatial Domain Research journal of Applied Science, Engineering and technology 4(23); 5063-5067, ISSN; 2040-7467, 2012

Ulman, M. C. Z. D. and. Zurada, J. M.(1993). Effective RNS scaling algorithm with the Chinese remainder theorem decomposition, in Proc. IEEE Pacific Rim Conf. Commune., Computer, Signal Process., Victoria, BC, Canada, May 1993, pp. 528–531.

Weyori, B.A. Amponsah, P.N and Yeboah P.K (2012). Modeling a secured digital image encryption using three moduli set. Global Journal of Computer Science and Technology Interdisciplinary, Vol.12 Issue 10 Version 10

Yang J. H. Chang C. C and Chen Y. Y (2004). A High Speed Division Algorithm in RNS using the Parity Checking Technique. International Journal of ComputerMathematics.**81**(6):775-780.

APPENDICES

APPENDIX A

The encryption codes Programmed in C++

```
#include <iostream>  #include <stdio.h> #include <math.h> #include <time.h>using namespace std;

/* run this program using the console pauser or add your own getch, system("pause") or input loop
*/ int mdc,mdc1,mdc2,alpha,beta,beta1,alpha1,alpha2,beta2; int modular(int base, unsigned intexp,
unsigned int mod) { int x = 1; inti; int power = base % mod; for (i = 0; i<sizeof(int) * 8; i++) {

intleast_sig_bit = 0x00000001 & (exp>>i); if (least_sig_bit)  x = (x * power) % mod; power = (power *
power) % mod; } return x;}

//chinese Rem Theory

intmul_inv(int an, intbn) {int b0 = bn, t, q; int x0 = 0, x1 = 1;

if (bn == 1) return 1; while (an > 1) { q = an / bn; t = bn, bn = an % bn, an = t;t= x0, x0 = x1 - q * x0, x1
= t; } if (x1 < 0) x1 += b0; return x1;} intchinese_remainder(int *n, int *an, intlen)

{int p, i, prod = 1, sum = 0; for (i = 0; i<len; i++) prod *= n[i];

for (i = 0; i<len; i++) {p = prod / n[i]; sum += an[i] * mul_inv(p, n[i]) * p;}

return sum % prod;} //int m1,m2,m3,q,a,v,s,d,v1,s1,d1,b,n;int main() {clock_t start = clock();

int
a,b,b1,b2,k,k1,k2,q1,q2,q3,q4,q5,q6,x1,x2,x3,x4,x5,x6,y1,y2,y3,y4,y5,y6,q,r,nn,n1,n2,v,s,z,d,v1,s1,d1,
m1,m2,m3,temp,temp1,temp2,j,j1,j2,w,e,u,o,o1,o2;

int chM,ch1,ch2,ch3,how,now,gan,fad,van,san,dan,crtt,now1;

printf("\nInput moduli set m1: "); scanf("%d",&b); printf("\nInput moduli set m1 again: ");
scanf("%d",&k); printf("\nInput moduli set m2: "); scanf("%d",&b1);

printf("\nInput moduli set m2 again: "); scanf("%d",&k1); printf("\nInput moduli set m3: ");
scanf("%d",&b2);

printf("\nInput moduli set m3 again: "); scanf("%d",&k2); //       j=b; j1=b1; j2=b2;

printf("\nEnter X:"); scanf("%d",&q); printf("\nEnter Y:"); scanf("%d",&z); printf("\nEnter N:");
scanf("%d",&now); printf("\nEnter N:"); scanf("%d",&now1); printf("\nEnter b:");
scanf("%d",&how);

v = q % b; s = q % b1; d = q % b2; a = z % b; s1 = z % b1; d1 = z % b2;

cout<< "X expressed in RNS = ("<< v <<","<<s <<","<< d<<")" <<endl;
```

```
cout<< "Y expressed in RNS = ("<< a <<","<<s1 <<","<< d1<<")" <<endl;

//x mod y

r=q%z; m1=r%b; m2=r%b1; m3=r%b2;

cout<< "R =:"<< r <<endl;

cout<< "R expressed in RNS = ("<< m1 <<","<<m2 <<","<< m3<<")" <<endl;

//X-R

w=v-m1; e= s-m2; u=d-m3; nn=w%b; n1=e%b1; n2= u%b2;

if(b>a) // fix if not a>b

temp=a; a=b ; b=temp ; x1=1 ; y1=0; x2=0 ; y2=1; while(a!=0) {q1=a/b ;a=a%b ;

if(a==0) { // Last Line,end of the algorithm. mdc=b; alpha=x2; beta=y2 ; break; // copy the values}

q2=b/a; b=b%a ; x1=x1-q1*x2; y1=y1-q1*y2; x2=x2-q2*x1; y2=y2-q2*y1; if(b==0) {// Last Line,end of the algorithm.

mdc=a ; alpha=x1; beta=y1 ; break; // copy the values} }

printf("gcd(a,b) = %d ",mdc); printf("\nalpha=%d ,beta=%d",alpha,beta);

cout<<endl;

if (beta<0){beta =k+beta; cout<<"beta"<<beta<<endl;} //beta2 if(b1>s1) // fix if not a>b

temp1=s1; s1=b1 ; b1=temp1 ; x3=1 ; y3=0; x4=0 ; y4=1; while(s1!=0) { q3=s1/b1 ; s1=s1%b1 ;

if(s1==0) { // Last Line,end of the algorithm.  mdc1=b1; alpha1=x4; beta1=y4 ; break; // copy the values}

q4=b1/s1;b1=b1%s1 ;x3=x3-q3*x4;y3=y3-q3*y4;x4=x4-q4*x3;y4=y4-q4*y3;

if(b1==0) {// Last Line,end of the algorithm. mdc1=s1 ; alpha1=x3; beta1=y3 ; break; // copy the values} } /*if (beta1<0){beta1 =k1+beta1;cout<<"k1= "<<k1<<endl;}*/

printf("gcd(a1,b1) = %d ",mdc1); printf("\nalpha1=%d ,beta1=%d",alpha1,beta1);

cout<<endl; //printf("\nEnter k1:"); scanf("%d",&k1);

if (beta1<0){beta1 =k1+beta1;cout<<"Beta= "<<beta1<<endl;} //beta3

if(b2>d1) // fix if not a>b temp2=d1; d1=b2 ; b2=temp2 ; x5=1 ; y5=0; x6=0 ; y6=1;

while(d1!=0) {q5=d1/b2 ;d1=d1%b2 ;if(d1==0) { // Last Line,end of the algorithm.

mdc2=b2; alpha2=x6; beta2=y6 ; break; // copy the values}

q6=b2/d1;b2=b2%d1 ;x5=x5-q5*x6; y5=y5-q5*y6; x6=x6-q6*x5; y6=y6-q6*y5;
```

```
if(b2==0) {// Last Line,end of the algorithm.

mdc2=d1 ; alpha2=x5; beta2=y5 ; break; // copy the values} }

printf("gcd(a2,b2) = %d ",mdc2); printf("\nalpha2=%d ,beta2=%d",alpha2,beta2);

cout<<endl; if (beta2<0){beta2 =k2+beta2; cout<<"beta2= "<<beta2<<endl;}

cout<< "Y^-1= : = ("<< beta <<","<<beta1 <<","<< beta2<<")" <<endl;

o=nn*beta; o1=n1*beta1; o2= n2 * beta2;

j=o%k; j1=o1%k1; j2=o2%k2; cout<< "(X-R): = ("<<nn<<","<<n1 <<","<< n2<<")" <<endl;

cout<< "(X-R)*Y^-1: = ("<< o <<","<<o1 <<","<< o2<<")" <<endl;

cout<< "Q=((X-R)*Y^-1): = ("<< j <<","<<j1 <<","<< j2<<")" <<endl;

if (j== 0){j=2;} else if (j1==0){j1=2;} else if (j2==0){j2=2;}

//chinese remainder theory;

chM= k*k1*k2;ch1=chM/k;ch2=chM/k1;ch3=chM/k2;

//C R T intcrt; int f[3]; f[0]=k; f[1]=k1; f[2]=k2; int t[3]; t[0]=j; t[1]=j1; t[2]=j2;int n[] = { k, k1, k2 };

int an[] = { j, j1, j2 };printf("\n Chinese Remainder =%d", chinese_remainder(n, an, sizeof(n)/sizeof(n[0]))); crt=chinese_remainder(n, an, sizeof(n)/sizeof(n[0])); //cout<<crt<<endl; if (crt> now){ crtt =crt % now1; cout<<" Since CRT is > n expect "<<crtt<<endl;} printf("\n Y ^ b mod n =%d", modular(crt, how, now)); fad=modular(crt, how, now); //cout<<fad<<endl;

van = modular(crt, how, now) % k;san = modular(crt, how, now) % k1;dan = modular(crt, how, now) % k2;cout<< " G expressed in RNS = ("<< van <<","<<san <<","<<dan<<")" <<endl;

clock_t stop = clock(); double elapsed = (double)(stop - start) * 1000.0 / CLOCKS_PER_SEC;printf("Time elapsed in ms: %f", elapsed); return 0;}
```

APPENDIX B

The Decryption Codes Programmed in C++

```
#include <stdio.h> #include <iostream> #include <math.h> #include <time.h> using namespace std;

/* run this program using the console pauser or add your own getch, system("pause") or input loop
*/int modular(int base, unsigned intexp, unsigned int mod) {int x = 1;inti; int power = base % mod;

for (i = 0; i<sizeof(int) * 8; i++) {intleast_sig_bit = 0x00000001 & (exp>>i); if (least_sig_bit)

 x = (x * power) % mod; power = (power * power) % mod; }

return x;} intmul_inv(int an, intbn){int b0 = bn, t, q; int x0 = 0, x1 = 1;if (bn == 1) return 1;

while (an > 1) {q = an / bn; t = bn, bn = an % bn, an = t; t = x0, x0 = x1 - q * x0, x1 = t;}

if (x1 < 0) x1 += b0; return x1;}intchinese_remainder(int *n, int *an, intlen){int p, i, prod = 1, sum = 0;

for (i = 0; i<len; i++) prod *= n[i]; for (i = 0; i<len; i++) {p = prod / n[i];sum += an[i] * mul_inv(p, n[i]) *
p;} return sum % prod;} int main() {

clock_t start = clock(); int Q,x,y,m1,m2,m3,v,s,d,v1,s1,d1,l,nnn,f1,b,e,t,h,van,dan,san; cout<<"input t
:"; cin>> t; cout<<"input f :"; cin>> f1; cout<<"input h :"; cin>> h; cout<<"Input moduli set m1:
";cin>>m1; cout<<"Input moduli set m2: ";cin>>m2; cout<<"Input moduli set m3: ";cin>>m3;

intcrt; int f[3];f[0]=m1; f[1]=m2; f[2]=m3; int t2[3]; t2[0]=t; t2[1]=f1; t2[2]=h; int n[] = { m1, m2, m3 };

int an[]={t,f1,h};printf("%d\nChinese Remainder= :", chinese_remainder(n, an,
sizeof(n)/sizeof(n[0]))); crt=chinese_remainder(n, an, sizeof(n)/sizeof(n[0]));

cout<<crt<<endl; cout<< "input n :";cin>>nnn; cout<<" input a secret keys: " ;cin>>b;

printf("\n Y^b mod n =%d ", modular(crt, b, nnn));

Q=modular(crt, b, nnn);//cout<<Q<<endl; van = modular(crt, b, nnn) % m1;

san = modular(crt, b, nnn) % m2; dan = modular(crt, b, nnn) % m3;

cout<< "G expressed in RNS = ("<< van <<","<<san <<","<<dan<<")" <<endl;

        clock_t stop = clock();

double elapsed = (double)(stop - start) * 1000.0 / CLOCKS_PER_SEC;

printf("Time elapsed in ms: %f", elapsed);return 0;}
```

APPENDIX C

Codes for Chin, Y. H. and Behrooz, P. (1994) Algorithms

```
#include <iostream> #include <stdio.h> #include <math.h> #include <time.h> using namespace std;

/* run this program using the console pauser or add your own getch, system("pause") or input loop
*/int mdc,mdc1,mdc2,alpha,beta,beta1,alpha1,alpha2,beta2;int modular(int base, unsigned intexp,
unsigned int mod) {int x = 1; inti; int power = base % mod; for (i = 0; i<sizeof(int) * 8; i++)
{intleast_sig_bit = 0x00000001 & (exp>>i); if (least_sig_bit)  x = (x * power) % mod;

power = (power * power) % mod;  }return x;}//chinese Rem Theory

intmul_inv(int an, intbn)

{int b0 = bn, t, q; int x0 = 0, x1 = 1; if (bn == 1) return 1; while (an > 1) {q = an / bn; t = bn, bn = an %
bn, an = t;t = x0, x0 = x1 - q * x0, x1 = t;}

if (x1 < 0) x1 += b0; return x1;} intchinese_remainder(int *n, int *an, intlen) {int p, i, prod = 1, sum =
0;for (i = 0; i<len; i++) prod *= n[i]; for (i = 0; i<len; i++) {

p = prod / n[i]; sum += an[i] * mul_inv(p, n[i]) * p;} return sum % prod;}int main() {clock_t start =
clock(); //clock_tstartTime = clock();

int
a,a1,a2,a3,a4,a11,b,b1,b2,k,k1,k2,q1,q2,q3,q4,q5,q6,x1,x2,x3,x4,x5,x6,y1,y2,y3,y4,y5,y6,r,nn,n1,n2,v
,s,z,d,v1,s1,d1 ,M1,M2,M3,m1,m2,m3;

int
chM,chM1,chM2,ch1,ch2,ch3,how,now,gan,fad,van,san,dan,BigM,Dinv,BZ,beta7,BZ1,BZ2,BZ3,ss1,kk
1,kk2,KK3,BigY,modf,mods,modd  ;

int
q,x,lastx,last1x,last2x,y,lasty,last1y,last2y,temp,temp1,temp2,temp3,tempx,tempx1,tempx2,tempx3,
bbZ1,bbz1,bbz2,bbz3,aaa;//modula set

printf("\nInput moduli set m1: "); scanf("%d",&b); printf("\nInput moduli set m1 again: ");
scanf("%d",&k); printf("\nInput moduli set m2: "); scanf("%d",&b1);

printf("\nInput moduli set m2 again: "); scanf("%d",&k1); printf("\nInput moduli set m3: ");
scanf("%d",&b2);printf("\nInput moduli set m3 again: "); scanf("%d",&k2);

printf("\nEnter  X:"); scanf("%d",&fad);printf("\nEnter  Y:"); scanf("%d",&a);printf("\nEnter  Y
again:"); scanf("%d",&a3);printf("\nEnter  Y again:"); scanf("%d",&s1);

printf("\nEnter  Y again:"); scanf("%d",&aaa);printf("\nEnter  n :"); scanf("%d",&gan);printf("\nEnter
b Public Key  :"); scanf("%d",&dan);//for //BigM =M and BZ=Z BigM=b*b1*b2;BZ=BigM%a;

M1=BigM/k;m1=M1%k;M2=BigM/k1;m2=M2%k1; M3=BigM/k2;m3=M3%k2; now=a-BZ;

cout<<"BigM = "<<BigM<<endl; cout<<"BZ = "<<BZ <<endl; //Dinv==
```

v = fad % b; s = fad % b1; d = fad % b2; a11 = a % b; ss1 = a % b1;d1 = a % b2;

cout<< "X expressed in RNS = ("<< v <<","<<s <<","<< d<<")" <<endl;

cout<< "Y expressed in RNS = ("<< a11 <<","<<ss1 <<","<< d1<<")" <<endl;//gcdif (b>a) {//we switch them temp=a; a=b; b=temp; } //begin function x=0;y=1;lastx=1; lasty=0; while (b!=0) { q= a/b;

temp1= a%b; a=b; b=temp1;temp2=x; x=lastx-q*x; lastx=temp2; temp3=y; y=lasty-q*y;

lasty=temp3; } if (lastx<0){lastx=lastx+k;}cout<< "gcd =:" << a <<endl; cout<< "x=" <<lastx<<endl;

// cout<< "y=" <<lasty<<endl; int Bk1=-BZ*lastx; kk1=Bk1%k;if (kk1<0){

kk1=kk1+k;} cout<< "K1=" << kk1 <<endl;bbz1=kk1*aaa;BZ1=(bbz1+BZ)/k;cout<< "Z1=" << BZ1 <<endl;//cout<<" a1, b1 "<<s1<<b1;//SECOND if(b1>s1) // fix if not a>b temp1=s1; s1=b1 ; b1=temp1 ;

x3=1 ; y3=0;x4=0 ; y4=1;while(s1!=0) {q3=s1/b1 ;s1=s1%b1 ;if(s1==0) { // Last Line,end of the algorithm. mdc1=b1; alpha1=x4; beta1=y4 ; break; // copy the values}

q4=b1/s1;b1=b1%s1 ;x3=x3-q3*x4;y3=y3-q3*y4;x4=x4-q4*x3;y4=y4-q4*y3;

if(b1==0) {// Last Line,end of the algorithm. mdc1=s1 ; alpha1=x3; beta1=y3 ; break; // copy the values}if (beta1<0){beta1 =k1+beta1;cout<<"k1= "<<k1<<endl;}//printf("gcd(a1,b1) = %d ",mdc1);

printf("\nalpha1=%d ,beta1=%d",alpha1,beta1); int Bk2=-BZ*beta1;kk2=Bk2%k1;if (kk2<0){

kk2=kk2+k1;}cout<< "Kk1=" << kk2 <<endl; bbz2=kk2*aaa; BZ2=(bbz2+BZ)/k1;

cout<< "Z2=" << BZ2 <<endl;

//beta3 if(b2>d1) // fix if not a>b temp2=d1; d1=b2 ; b2=temp2 ;

x5=1 ; y5=0;x6=0 ; y6=1; while(d1!=0) {q5=d1/b2 ;d1=d1%b2 ;

if(d1==0) { // Last Line,end of the algorithm. mdc2=b2; alpha2=x6; beta2=y6 ; break; // copy the values}q6=b2/d1;b2=b2%d1 ;x5=x5-q5*x6;y5=y5-q5*y6;x6=x6-q6*x5;y6=y6-q6*y5;

if(b2==0) {// Last Line,end of the algorithm. mdc2=d1 ; alpha2=x5; beta2=y5 ; break; // copy the values}}//printf("gcd(a2,b2) = %d ",mdc2);printf("\nalpha2=%d ,beta2=%d",alpha2,beta2);

cout<<endl; if (beta2<0){beta2 =k2+beta2;cout<<" beta2= "<<beta2<<endl;}

int Bk3=-BZ*beta2; int kk3=Bk3%k2;if (kk3<0){kk3=kk3+k2;}cout<< " Kk3= " << kk3 <<endl;

bbz3=kk3*aaa;BZ3=(bbz3+BZ)/k2; cout<< "Z3=" << BZ3 <<endl;

intcrt; int f[3]; f[0]=k; f[1]=k1;f[2]=k2;int t2[3];

t2[0]=m1; t2[1]=m2; t2[2]=m3;int n[] = { k, k1, k2 };int an[] = { m1, m2, m3 };

printf("\nChinese Remainder= : %d", chinese_remainder(n, an, sizeof(n)/sizeof(n[0])));

```
crt=chinese_remainder(n, an, sizeof(n)/sizeof(n[0]));how=crt%693;

int BX=crt-how; int BZZ=BZ1+BZ2+BZ3;ch1=(v*lastx)%k; ch2=(s*beta1)%k1; ch3=(d*beta2)%k2;int add=ch3+ch1+ch2; chM=add*BZ1;chM1=add*BZ2;chM2=add*BZ3;int add1=chM+chM1+chM2;

BigY=add1-(BX*now);//cout<<how <<endl; cout<<"B(X) = "<<BX <<endl; int fine=fad%aaa; if (fine !=0){cout<<"Y = "<<BigY<<endl; printf("%d\nY^b mod n = ", modular(BigY, dan, gan));

fad=modular(BigY, dan, gan); modf=fad%k; mods=fad%k1;modd=fad%k2;

cout<< "Y expressed in RNS = ("<<modf<<","<<mods <<","<<modd<<")" <<endl; }else{ nn=(fad-fine )*aaa; r=nn%BigM;} clock_t stop = clock(); double elapsed = (double)(stop - start) * 1000.0 / CLOCKS_PER_SEC;

double elapsed1=pow(elapsed,2); printf("Time elapsed in ms: %f", elapsed); printf("Time elapsed in ms: %f", elapsed1);//cout<<double( clock() - startTime ) / (double)CLOCKS_PER_SEC<< " seconds." <<endl;cout<<"try again"<<endl; cin>>gan; if (gan==1){ goto start;}else{return 0;}}
```