# Improvement in Performance of RSA Cryptosystem

**A Thesis**

***submitted in partial fulfilment for the award of the degree of***

***Doctor of Philosophy***


***submitted by***

**Seema Verma**

**(Reg. No. 950803017)**

***Under the guidance of***

**Dr. Deepak Garg**

**(Associate Professor & Head)**

**Department of Computer Science & Engineering**

**Thapar University**

**Patiala-147004, Punjab, India**


**July, 2015**

*Vakratunda Mahakaya Suryakoti Samaprabha*
*Nirvighnam Kuru Me Deva Sarva-Kaaryessu Sarvadaa*

*Saraswati Namasthubhyam Varade Kamarupini*
*Vidhyarambham Karishyami Siddhir Bavathume Sadha*

*Dedicated to*

*My Family*

*My Mother-in-Law*

*My Parents*

# *Certificate*

This is to certify that thesis entitled "Improvement in Performance of RSA Cryptosystem" submitted by Seema Verma to Thapar University, Patiala, for the award of the degree of "Doctor of Philosophy" is a bonafide record of the research work carried out by her under my supervision and guidance. The content of the thesis, in full or parts have not been submitted to any other Institute or University for the award of any other degree or diploma.

Seema Verma
950803017

Research Supervisor

Dr. Deepak Garg
Associate Professor & Head
Department of Computer Science and Engineering
Thapar University, Patiala

Date: 27th July, 2015

# *Abstract*

Cryptography is used to communicate the critical information over an insecure network. Symmetric key cryptographic algorithms are used to encrypt the information so that the intruder cannot get the information of the original message. These algorithms are fast due to small (secret) key size, say 128 bits. As the secret key between the sender and receiver is communicated over insecure channel, one cannot use symmetric key algorithms for complete communication. Shared key can be compromised over insecure network. Public key cryptographic algorithms are needed at least to communicate the shared key used by the symmetric key algorithms. In public key algorithms, key used is very large to secure the message to be communicated. The large sized key comes at its own cost. The key size affects the computational performance of the cryptosystem. Also, due to the large key size, it creates storage problem. In resource constrained devices, large key size in public key algorithms becomes bottleneck for the performance.

RSA is the most popular and widely studied public key algorithm. As with the other public key algorithms, it also requires very large key size. This drawback of RSA prompts one to shift from this public key cryptosystem to other having smaller key size for resource constrained devices. Also, the use of other public key cryptosystems with smaller key size comes with their own shortcomings. Hence there is a need to improve RSA cryptosystem so that it results in better efficiency when used in resource constrained devices. This thesis proposes methods to improve the computational efficiency and memory consumption for different RSA variants.

Thesis includes the study of different existing RSA variants. The variants are studied in terms of their encryption and decryption efficiency. Some of the variants work on encryption performance and some on decryption performance. Based on the computational efficiency, comparative analysis of these variants is carried out to better understand the performance of RSA variants. Memory consumption of different variants is also studied.

One of the variant, Rebalanced RSA CRT Scheme (RRCS), can be used in any scenario where the user requires selection among the gain in encryption or decryption performance. In this thesis, work is done to improve the decryption speed of RRCS by using multiple primes. While maintaining the same encryption speed, the decryption speed of RRCS increases by a factor of 1.8. Implementation of the scheme is shown to reflect the performance gain.

Dual RSA (another variant of RSA cryptosystem) is studied and analyzed for further improvement. Three schemes of Dual RSA are improved to design three new schemes; i.e. RC RSA-I, RC RSA-II and RC RSA-III. In RC RSA-I, multiple primes are used in Dual RSA Small-e to improve the decryption performance. In RC RSA-I the computational performance as well as memory consumption improves as compared to Dual RSA Small-e. In RC RSA-II, slow computations involved in Dual RSA Small-d are performed offline when the computing device is relatively free; this improves the on-line encryption performance. In RC RSA-III, the use of multiple primes in Dual Generalized Rebalanced RSA improves the decryption performance. These three schemes are implemented and compared to reflect the better results.

# *Acknowledgements*

During my entire tenure as a student of Ph.D, I was inspired and supported by many individuals who helped me achieve this goal.

I am thankful to my research guide and guru Dr. Deepak Garg who inspired me and showed me the light in the darkest of the time of this journey right from the beginning. It was quite impossible to achieve this target without his continuous support at every step. No words can ever be enough for untiring support and guidance I got from him.

I am highly thankful to the staff of Computer Science and Engineering department of Thapar University, Patiala who helped in all the related matters during my every visit to the Institute.

I am deeply thankful to the doctoral committee members of Thapar University for their valuable suggestions and guidance.

I am also thankful to my colleagues and friends at Echelon Institute of Technology, Faridabad for their valuable support.

I am thankful to all the anonymous reviewers of my work who helped me greatly improve my work by their valuable suggestions.

I am thankful to my family for their patience at all the time during my research tenure; especially I cannot forget the support of my husband as he always stood by me to realize my dream.

At last but not the least, I am indebted to Almighty supreme power who gave me enough courage to take this challenging work.

SEEMA VERMA

# Contents

**Abstract** iv

**Acknowledgements** vi

**List of Figures** x

**List of Tables** xi

**1 Introduction** 1
- 1.1 Cryptography . . . 1
  - 1.1.1 Security Goals . . . 2
  - 1.1.2 Types of Cryptographic Systems . . . 3
    - 1.1.2.1 Private Key Cryptosystem . . . 3
    - 1.1.2.2 Public Key Cryptosystem . . . 4
- 1.2 Motivation . . . 5
- 1.3 Contribution of the Thesis . . . 5
- 1.4 An Overview of the Thesis . . . 6

**2 Analysis of RSA and its Variants** 7
- 2.1 Mathematical Background . . . 8
  - 2.1.1 Integer Arithmetic . . . 8
  - 2.1.2 Modular Arithmetic . . . 9
  - 2.1.3 Prime Numbers . . . 11
  - 2.1.4 Primality Testing . . . 12
  - 2.1.5 Factorization Methods . . . 13
  - 2.1.6 Chinese Remainder Theorem . . . 15
  - 2.1.7 Continued Fractions . . . 17
  - 2.1.8 Lattice Theory . . . 18
- 2.2 Standard RSA . . . 19
  - 2.2.1 Efficiency of RSA operations . . . 20
  - 2.2.2 Attacks on RSA Cryptosystem . . . 21
    - 2.2.2.1 Elementary Attacks . . . 21
    - 2.2.2.2 Small-$e$ Attack . . . 22

2.2.2.3 Small-$d$ Attack . . . 23
2.2.2.4 Partial Key Exposure Attack . . . 25
2.2.3 Improvement in Standard RSA . . . 26
2.3 Basic Variants (Improvement in Decryption Performance) . . . 27
2.3.1 RSA CRT (Chinese Remainder Theorem) . . . 28
2.3.1.1 Efficiency . . . 29
2.3.1.2 Attacks . . . 29
2.3.2 Batch RSA . . . 30
2.3.2.1 Efficiency . . . 31
2.3.3 MultiPrime RSA . . . 32
2.3.3.1 Efficiency . . . 33
2.3.3.2 Attacks . . . 34
2.3.4 MultiPower RSA . . . 35
2.3.4.1 Efficiency . . . 36
2.3.4.2 Attacks . . . 37
2.3.5 Rebalanced RSA . . . 38
2.3.5.1 Efficiency . . . 38
2.3.5.2 Attacks . . . 39
2.4 Balancing of Encryption/Decryption Performance . . . 40
2.4.1 Rebalanced RSA CRT Scheme (RRCS) . . . 40
2.4.1.1 Efficiency . . . 41
2.4.1.2 Attacks . . . 42
2.5 Improvement in Memory Consumption . . . 43
2.5.1 Dual RSA . . . 43
2.5.1.1 Memory Usage . . . 45
2.5.1.2 Attacks . . . 45
2.6 Comparison of RSA Variants . . . 46

**3 Improvement in Rebalanced RSA CRT Scheme** **50**
3.1 Extended Rebalanced RSA CRT Scheme (ERRCS) . . . 50
3.1.1 Key Generation Method . . . 51
3.1.2 Encryption and Decryption Method . . . 52
3.2 Security Analysis . . . 53
3.2.1 Factorization Attack . . . 53
3.2.2 Parameter Selection Attack (Lattice method) . . . 54
3.2.3 Security Summary . . . 58
3.3 Complexity Analysis . . . 59
3.4 Implementation Details . . . 61
3.5 Comparative Analysis . . . 64
3.6 Conclusion . . . 67

**4 Improvement in Dual RSA** **69**
4.1 RC RSA-I . . . 70
4.1.1 Key Generation Method . . . 70

4.1.2 Encryption and Decryption Method . . . 72
4.1.3 Security Analysis . . . 73
4.1.4 Complexity Analysis . . . 76
4.1.5 Implementation Details . . . 78
4.2 RC RSA-II . . . 80
4.2.1 Key Generation Method . . . 81
4.2.2 Encryption and Decryption Method . . . 81
4.2.3 Security Analysis . . . 82
4.2.4 Complexity Analysis . . . 83
4.2.5 Implementation Details . . . 85
4.3 RC RSA-III . . . 87
4.3.1 Key Generation Method . . . 87
4.3.2 Encryption and Decryption Method . . . 89
4.3.3 Security Analysis . . . 89
4.3.4 Complexity Analysis . . . 97
4.3.5 Implementation Details . . . 98
4.4 Comparative Analysis . . . 103
4.5 Conclusion . . . 108

**5 Conclusions and Future Work 109**
5.1 Conclusions . . . 109
5.1.1 Improvement in Rebalanced RSA CRT Scheme . . . 109
5.1.2 Improvement in Dual RSA . . . 110
5.2 Future Directions . . . 111

**A NTL (A Library for Number Theory) 113**

**B Examples based on Proposed Schemes 116**
B.1 ERRCS(Chapter 3) . . . 116
B.2 RC RSA-I (Chapter 4) . . . 122
B.3 RC RSA-II (Chapter 4) . . . 125
B.4 RC RSA-III (Chapter 4) . . . 128

**C Publications Based on the Thesis Work 138**

**References 139**

# List of Figures


1.1 Private Key Cryptosystem . . . . . 3
1.2 Public Key Cryptosystem . . . . . 4

2.1 Classification of RSA Variants . . . . . 27
2.2 Comparison of existing RSA Variants based on Computational Performance . . . . . 48

3.1 Complexity comparison of the Proposed Scheme with existing RSA Variants . . . . . 65
3.2 Comparison of the Proposed Scheme with existing RSA Variants: Implementation . . . . . 67

4.1 Comparison of RC RSA Schemes with existing RSA variants . . . . 105
4.2 Comparison of Memory Consumption by RC RSA Schemes with existing RSA variants . . . . . 106
4.3 Comparison of RC RSA Schemes with existing RSA variants (Implementation) . . . . . 107

# List of Tables


2.1 Example based on Extended Euclidean Algorithm . . . . . 9
2.2 Comparison of Existing RSA variants based on Computational Performance . . . . . 47
2.3 Comparison of existing RSA variants based on Memory Usage . . . 48

3.1 Implementation results of the proposed scheme for N=1024bits . . . 62
3.2 Implementation results of the proposed scheme for N=2048bits . . . 64
3.3 Complexity comparison of the Proposed Scheme with existing RSA Variants . . . . . 65
3.4 Comparison of the proposed scheme with existing RSA variants: implementation . . . . . 66
3.5 Comparison of proposed scheme with existing RSA variants implemented on cPCI Architecture . . . . . 67

4.1 Implementation Results of RC RSA-I for N=1024 bits . . . . . 79
4.2 Implementation Results of RC RSA-I for N=2048 bits . . . . . 80
4.3 Implementation Results of RC RSA-II for N=1024bits . . . . . 86
4.4 Implementation Results of RC RSA-II for N=2048 bits . . . . . 86
4.5 Implementation Results of RC RSA-III for N=1024 bits . . . . . 101
4.6 Implementation Results of RC RSA-III for N=2048 bits . . . . . 102
4.7 Complexity Comparison of the proposed schemes with existing RSA variants . . . . . 104
4.8 Comparison of the proposed schemes with existing RSA variants (Implementation) . . . . . 106
4.9 Comparison of proposed schemes with existing RSA variants implemented on cPCI Architecture . . . . . 107

A.1 NTL Routines based on Basic Arithmetic . . . . . 114
A.2 NTL Routines based on Modular Arithmetic . . . . . 115
A.3 NTL Routines based on Random Numbers . . . . . 115
A.4 NTL Routines based on Prime Numbers . . . . . 115

# Chapter 1

# Introduction

## 1.1 Cryptography

The need to secure crucial information exchanged over insecure network gave birth to the concept of cryptography. While practicing the cryptography, sender encrypts or encodes the information with a secret key in such a way that only intended recipient can understand it. On the other hand, cryptanalysis signifies the unwanted access to the information without the secret key. Cryptography and cryptanalysis combine to form the term 'Cryptology'.

The concept of cryptography is not new; its foundation can be found in hieroglyphic symbols by Egyptians in 1900 BC. But this writing was not used for the purpose of secrecy; rather it gives the original text a new form. Other form of cryptography can be seen in "Arthashastra" by Kautilya. In ancient history, cryptography was used by the Romans around 100 BC. In 1844, communication through telegraph made the cryptography stronger and desirable. In 1883, A. Kerckhoffs described few rules for the secure communication; the most widely known rule which is still applicable in today's scenario is that the method to secure the information (not the key) must be publicly known to everyone. This rule formed the basis for modern cryptography. Taking it further, in 1949, Claude Shannon (father of Information theory) gave the theory of perfect secrecy by introducing the concept of 'One Time Pad'. This theory states that the system is computationally secure even with the use of the strongest cryptanalysis method. After

Shannon's work, modern cryptography assumed the main stage in the field of information security and in 20th century, advance number theory found its place in cryptography and cryptanalysis.

The first commercial encryption standard, DES (Data Encryption Standard) [1], was published in 1975. Due to less security (small key size of 56-bit), it lost its popularity and AES (Advanced Encryption standard, primarily known as Rijndael) [2] evolved. In 1976, concept of public key was introduced by Whitfield Diffie and Martin Hellman [3]. First public key cryptographic algorithm, RSA [4], was designed in 1977. In late 1990s, due to the popularity of electronic transactions, the concept of hybrid approach was used. In hybrid approach, the message as a whole is encrypted by symmetric key algorithm, but the secret key used in the algorithm is transmitted using public key algorithm. Different encryption algorithms are in trend to secure the message in transit. These algorithms are designed to satisfy defined security goals.

### 1.1.1 Security Goals

Some security goals have to be achieved to make any system secure; modern cryptography is based on the following security goals:

- Confidentiality: To secure the information in such a way that it reaches to the intended recipient only.
- Integrity: To ensure that the contents that reaches to its intended recipient are not manipulated during transit through the communication channel.
- Authentication: To provide identity and authenticity to the information.
- Non-Repudiation: To ensure that the sender or receiver cannot deny their identification.

Any attempt to attack the cryptographic goals can threaten the security and integrity of the system. Any system can be intruded [6] and completely broken. Apart from above mentioned security goals, 'Access control' and 'Availability' are also important goals of cryptography. In access control, authorities are defined for a particular user. Availability means to limit the resources to the user. DoS (Denial of Service) attack can threaten the issue of availability. By this attack, resources can be made unavailable to the user [7], [8].

### 1.1.2 Types of Cryptographic Systems

In every cryptographic system, there is a need to secure the information by using some key. Modern cryptosystem can be classified on the basis of the number of keys used: private key cryptosystem and public key cryptosystem. Both of these cryptosystems are described briefly as follows:

#### 1.1.2.1 Private Key Cryptosystem

In private key cryptosystem (or symmetric key cryptosystem) (figure 1.1) both, the sender and receiver, share a single key. This key is used by the sender to encrypt the plain text to get the cipher text. The same key is used by the receiver to decrypt the cipher text to recover the plain text.

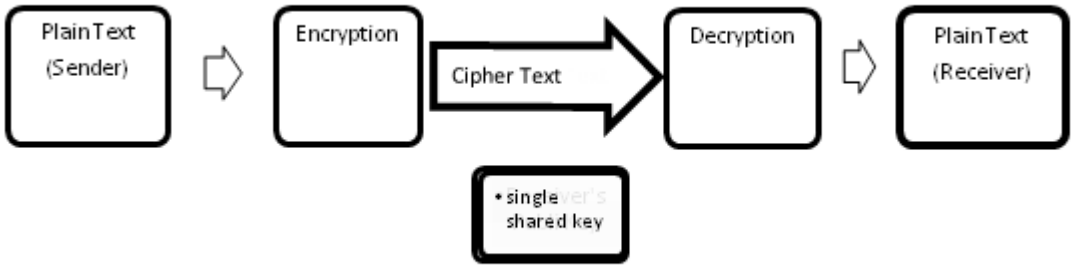


FIGURE 1.1: Private Key Cryptosystem

Some of the examples of symmetric key algorithms are DES [1], AES [2] etc. Generally all the symmetric key algorithms are fast, but the problem with this type of cryptosystem is about sharing the key among its users over the insecure communication channel. The solution to this problem is to use the public key, at least for sharing the secret key. As public key is very costly, it cannot be used to encrypt the complete message. Hybrid approach is used to overcome this problem, i.e. the message is encrypted using the symmetric key whereas the shared common key is encrypted with the public key.

#### 1.1.2.2 Public Key Cryptosystem

The concept of public key cryptosystem was introduced by W. Diffie and M. Hellman [3]. In the public key cryptosystem (figure 1.2), two keys are used, one key is publicly known to everyone whereas the other key is kept as secret. Public key is used to encode (encrypt) the message and the private key is used to decode (decrypt) the encoded message. All public key cryptosystems are based upon some one way functions (easy to compute but hard to inverse). Only the intended user can calculate the one way function by using some secret parameter (Trapdoor one way function). Legal users have the access to this secret parameter, but illegal users are not having access to the secret parameter. Product of two large prime numbers is assumed to be one way function. It is simple to compute the product of two large prime numbers, but finding the prime numbers from the given large composite integer is assumed to be hard. This problem is considered as the security of RSA [4], Rabin [9] etc. Other known public key cryptosystems are based on different types of hard problem; e.g. the problem of finding the discrete logarithm, which is used in Diffie-Hellman scheme (key exchange protocol), ElGamal etc. and problem of computation of elliptic curves as used in Elliptic curve cryptography.

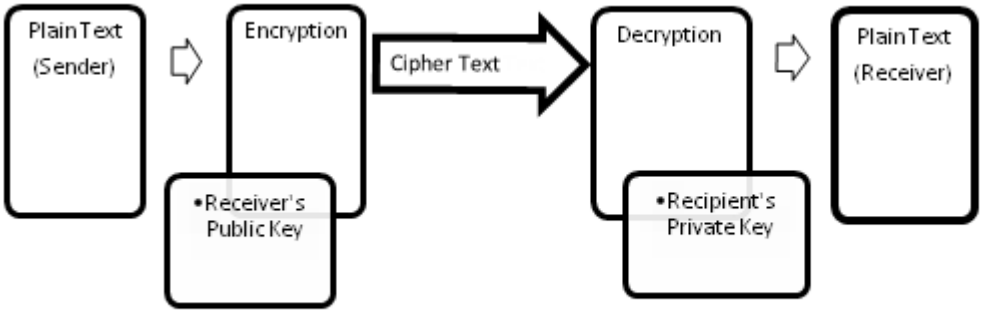


FIGURE 1.2: Public Key Cryptosystem

In RSA [4], public and private keys are calculated as inverse of each other. Private key can also be used by the sender to generate the digital signatures. Anyone who knows the public key of the sender can verify the message. If the message is intended for a particular receiver, the user applies the receiver's public key on the message. The message is accessed by the receiver; first by applying his/her own secret key and then sender's public key. Depending on the application, public and private key can be used in any order.

## 1.2 Motivation

Today's world is dependent on electronic communication. The major issue with such communication is to secure the critical information like financial information, passwords etc. In a rapidly shrinking world, communication is shifting from desktop to smart phones where resources are limited. Lots of cryptographic algorithms are designed to make the critical transactions secure. Among them, RSA has been the most popular cryptographic algorithm. The simplicity of this algorithm makes it more attractive to the security providers. Since the origin of RSA in 1977, it is widely used in almost every field of securing the information. Due to this very reason, RSA remains a popular interest among the researchers also. The security of this algorithm lies in computing the factors of a large composite integer. Currently 1024 bits (for non critical information) and 2048 bits or above (for critical information) composite integer is used to achieve desired security level. The workload of factoring a 1024 bit composite integer is assumed to be as complex as workload of $2^{80}$ instructions which is the current benchmark used in cryptography (likely to increase in future).

Due to the use of large key size, computations involved also get increased. The use of large key size becomes bottleneck if the system is working on small device or heavily loaded systems. Consequently the demand of the cryptosystem with fast computations arises.

Also to store the big keys involved in the cryptosystem, more storage is required. Thus, there is the need to minimize the storage requirement by RSA cryptosystem while simultaneously maintaining the acceptable security level.

## 1.3 Contribution of the Thesis

Variants of RSA cryptosystem are studied as part of this work. The efficiency of these variants is analyzed with the existing security constraints. Consequently the improvement is done in a few RSA variants so that they may be used in a resource constrained environment. Various RSA schemes are proposed to improve the computational efficiency and the memory usage by RSA cryptosystem.

Further the complexity of proposed schemes is analyzed and compared with the existing variants. It is proved that the proposed schemes are more efficient as compared to the existing variants.

To demonstrate the improved efficiency, proposed schemes are implemented using NTL with GMP on Cygwin tool for Windows operating system. Also, the proposed schemes are compared with the existing ones for efficiency.

## 1.4 An Overview of the Thesis

In this thesis, the improved schemes are proposed in RSA cryptosystem so that the computational efficiency and memory consumption gets better. The work is organized as follows:

Chapter 2 gives the analysis of the variants of RSA. The chapter starts with the mathematical background, followed by RSA cryptosystem with efficiency consideration and security attacks. Different RSA variants based upon their computational cost and memory consumption are discussed. These variants are analyzed in terms of efficiency and security aspects. Finally the chapter concludes with the comparison of different variants based on their computational efficiency and memory usage.

In Chapter 3, a new RSA scheme is proposed and the steps followed to achieve the improvement over existing RSA variant are elucidated. The proposed scheme is shown to have better decryption speed. The complexity of encryption and decryption is analyzed for computational performance.

Chapter 4 shows other improved schemes. Three schemes, namely RC RSA-I, RC RSA-II and RC RSA-III, are proposed to have improvement over existing RSA variant. In RC RSA-I, improvement is shown in decryption as well as encryption speed. In RC RSA-II, the on-line encryption speed is shown to be better. In RC RSA-III, work is shown to have better performance in decryption speed. All the schemes are analyzed theoretically and implemented for the comparative results.

In chapter 5, the important results are discussed conclusively and directions are laid for further work.

# Chapter 2

# Analysis of RSA and its Variants

The concept behind public key cryptography is to avoid the prior communication of the shared key. For this purpose two keys are used for communication; one key is publicly known to everyone (public key) whereas the other is kept as secret (private key). These keys should be computed such as to use one key for encryption (public key) and the other key for decryption (private key). RSA cryptosystem is based on computing the public and private keys as the modular inverse of each other. Modulus ($N$) is considered to be the product of two randomly chosen prime numbers ($p$ and $q$). For large modulus size, if one key is taken as small, the other results in large size. Size of the public and private keys (exponents) affects the performance of RSA cryptosystem. Performance of RSA cryptosystem is measured in terms of encryption speed, decryption speed or memory consumption. Encryption speed enhances with decrease in the bit size of the encryption exponent ($e$, public key) and decryption speed enhances with the decrease in bit size of decryption exponent ($d$, private key). Memory consumption by the cryptosystem reduces with the decrease in the bit size of parameters; encryption exponent ($e$), decryption exponent ($d$) or modulus ($N$). Standard RSA is based on small encryption exponent and large decryption exponent. Different variants are found in literature for the improvement in performance of RSA cryptosystem. Some of the variants are studied in following sections which gives sufficient background to the work presented in this thesis.

## 2.1 Mathematical Background

In this section, the required mathematical concepts [10] for RSA cryptosystem are briefly introduced. In general $n_x$ is taken to be the number of bits in any parameter $x$ in the work presented in this thesis.

### 2.1.1 Integer Arithmetic

Set of Non fractional numbers $[-\infty, \infty]$ forms the integer and is denoted by $\mathbb{Z}$. Addition, subtraction, multiplication and division are some of the binary operations performed on integers. In division if $y$ divides $x$; $q$ is the quotient and $r$ is the remainder. When $r$ is 0, $x$ is divisible by $y$ or $y|x$; e.g. 2|12, 4|20, 9|45.

**Greatest Common Divisor (GCD)**: $GCD(x, y) = m$, where $m$ is the largest number that can divide both $x$ and $y$; e.g. $GCD(34, 12) = 2$.

**Euclidean Algorithm**: This method is given by Euclid for the calculation of $GCD()$ of two numbers. The algorithm uses two facts; $GCD(x, 0) = x$ and $GCD(x, y) = GCD(y, r)$, here r is the remainder of $x/y$; example: $GCD(34, 12) = GCD(12, 10) = GCD(10, 2) = GCD(2, 0) = 2$ The relation between $LCM()$ (Least Common Multiple) and $GCD()$ is defined by the relation: $a \times b = GCD(a, b) \times LCM(a, b)$.

**Extended Euclidean Algorithm**: With the help of this algorithm linear Diophantine equations can be solved. The values $x$, $y$ and $m$ can be calculated for the equation $a_1 \times x + a_2 \times y = a_3$, where $m = GCD(a_1, a_2)$ . If $m$ does not divide $a_3$, one cannot find the integer values for $x$ and $y$, but if $m|a3$, there are infinitely many solutions for $x$ and $y$. For the solution, the equation can be rewritten as $k_1 \times s + k_2 \times t = k_3$, where $k_1 = a_1/m$, $k_2 = a_2/m$, $k_3 = a_3/m$ and $s$, $t$ are unknown variables. With the help of extended Euclidean algorithm one can find the value of $s$ and $t$.

Here is the example to calculate the values of $m, s, t$ for $161s + 28t = m$. In the example $r_1 = a_1 = 161$, $r_2 = a_2 = 28$, $s_1 = 1$, $s_2 = 0$; $t_1 = 0$, $t_2 = 1$, $q = r_1/r_2$. Variables are updated in each row as $r = r_1 - q * r_2$, $r_1 = r_2$, $r_2 = r$. In the same way $s$ (with $s_1$ and $s_2$) and $t$ (with $t_1$ and $t_2$) are updated till $r_2 = 0$:

From table, $m = r_1 = 7$, $s = s_1 = -1$ and $t = t_1 = 6$.

TABLE 2.1: Example based on Extended Euclidean Algorithm

| $q$ | $r_1, r_2$ | $r$ | $s_1, s_2$ | $s$ | $t_1, t_2$ | $t$ |
|---|---|---|---|---|---|---|
| 5 | 161, 28 | 21 | 1, 0 | 1 | 0, 1 | -5 |
| 1 | 28, 21 | 7 | 0, 1 | -1 | 1, -5 | 6 |
| 3 | 21, 7 | 0 | 1, -1 | 4 | -5, 6 | -23 |
| | 7, 0 | | -1, 4 | | 6, -23 | |

Particular solution for $x$ and $y$ can be calculated as:

$x_0 = (a_3/m) \times s$ and $y_0 = (a_3/m) \times t$

From this, all the values can be calculated as:

$x = x_0 + k(a_2/m)$ and $y = y_0 - k(a_1/m)$, (where $k$ is any integer)

### 2.1.2 Modular Arithmetic

Modular operator ( mod ) gives the remainder of two integers $a$ and $b$, when $a$ is divided by $b$. The result of ($a$ mod $b$) operator is a non negative integer value less than $b$. By mod operation, the set of least residues or $\mathbb{Z}_n$ is generated, which is the set of integer in the range $(0, n-1)$; e.g. $\mathbb{Z}_3 = \{0, 1, 2\}$ and $\mathbb{Z}_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. More than one number in $\mathbb{Z}$ can have the same value in $\mathbb{Z}_n$. For this representation instead of equality, congruence operator ($\equiv$) is used; e.g. $3 \equiv 13(\mod 10)$, $23 \equiv 93(\mod 10)$ and so on.

**Modular Operations**: Addition, subtraction and multiplication are defined in $\mathbb{Z}_n$. If two numbers have to be added, subtracted or multiplied, then the result of operation applies to mod operator, e.g. $(15 + 8) \mod 16 = 7$. Likewise the operations for subtraction and multiplications are performed.

**Modular Inverse**: Additive inverse in $\mathbb{Z}_n$ is $x+y \equiv 0(\mod n)$; e.g. in $\mathbb{Z}_{10}$, $(0, 0)$, $(1, 9)$, $(2, 8)$ and so on, six additive inverse exist. Multiplicative inverse is defined as $x * y \equiv 1(\mod n)$; e.g. in $\mathbb{Z}_{10}$, $(1, 1)$, $(3, 7)$ and $(9, 9)$ are pairs of multiplicative inverse. If $GCD(x, n) \equiv 1$ then the multiplicative inverse of $x$ is possible in $\mathbb{Z}_n$ and can be calculated by Extended Euclidean algorithm. All the numbers in $\mathbb{Z}_n$ are having additive inverse, but all the numbers in $\mathbb{Z}_n$ are not having multiplicative inverse. The subset of $\mathbb{Z}_n$ such that every number has the multiplicative inverse is denoted as $(\mathbb{Z}_n)^*$ (the set in which every member is relatively prime to n); e.g.

$(\mathbb{Z}_6)^* = 1, 5$. All the numbers in $(\mathbb{Z}_n)^*$ are not having additive inverse. In $(\mathbb{Z}_p)^*$, ($p$ is prime ), all the numbers are having additive as well as multiplicative inverse; e.g. $\mathbb{Z}_5 = \{0, 1, 2, 3, 4\}$, $(\mathbb{Z}_5)^* = \{1, 2, 3, 4\}$.

**Modular Exponentiation**: For modular exponentiation of a very large number, a special method of 'Square and Multiply' is used. Following is the method to calculate $a = x^b \mod c$; here $n_b$ is the number of bits in exponent $b$.

**Square and Multiply Method**:

**Algo SnM** $(x, b, c)$

{

$a = 1$

for($j = 0$ to $n_b - 1$)

{

if($b_i = 1$)

$a = x * a \mod c$

$x = x^2 \mod c$

}

return $a$

}

In this method, square operation is done for every bit in $b$ and multiply operation is done for every 1 bit present in $b$. For random exponent $b$, number of 1's is considered as $n_b/2$. Hence the complexity for exponentiation ($x^b \mod c$) is the sum of the complexity of $n_b$ times 'squares' and $n_b/2$ times 'multiply' modulo $c$; i.e. $(n_b + n_b/2)n_c^2$ or $1.5n_bn_c^2$, where $n_c$ is the bit size of the modulus $c$. Here the complexity of modular multiplication is assumed to be $O(n_c^2)$. Using Karatsuba method, multiplication complexity comes out to be of the order of $(n_c)^{1.59}$.

### 2.1.3 Prime Numbers

A number is prime if it is divisible by only 1 and itself. There are infinite number of primes. If a number is divisible by any number other than 1 and itself, then it is a composite number. If $GCD(x, y) = 1$, then the numbers $x$ and $y$ are coprime or relatively prime to each other.

**Prime number Theorem**: This theorem describes the distribution of prime number in the given range. Prime number theorem defines the complexity of generating the random prime number $x$ (size $n_x$ bit) as $O(n_x)$, where $n_x$ is the number of bits in $x$. Approximate number of primes ($\pi(x)$) can be calculated for any number $x$. Maximum and minimum number of primes are given by Gauss and Lagrange respectively. The range for number of primes is $\frac{x}{\ln x} < \pi(x) < \frac{x}{\ln x - 1.08366}$.

**Prime number generation**: One can find all the primes less than $x$ by the sieve of Eratosthenes. All the numbers between 2 to $x$ which are not divided by any prime less than $\sqrt{x}$, are prime numbers in the range of $(1, x)$. Prime numbers of type $X_x = 2^x - 1$ are called Mersenne primes; here $x$ is a prime number; e.g. $M_2 = 3$, $M_3 = 7$, $M_5 = 31$ and so on. But all numbers of this type are not prime; e.g. $M_{11} = 2047$ is not a prime. The prime numbers of type, $X_x = 2^{2^x} + 1$ are called Fermat primes. Fermat primes are defined only for $x < 5$; $X_0 = 3$, $X_1 = 5$, $X_2 = 17$, $X_3 = 257$, $X_4 = 65537$. $X_5$ is not a valid prime. If $X$ and $2X + 1$ are prime then $X$ is Sophie Germain prime, e.g. 29.

**Euler's Totient Function** ($\phi$):$\phi(x)$ is the number of integers less than and relatively prime to $x$. If $x$ is a prime number then $\phi(x)$ is $x - 1$. For two relatively prime numbers $x$ and $y$, $\phi(x * y)$ can be calculated by $\phi(x) * \phi(y)$. For prime number $x$, $\phi(x^{x_1})$ can be calculated as $(x^{x_1} - x^{x_1 - 1})$. One can find the totient function for a large composite number $x$, if $x$ can be factored into prime numbers. For $n > 2$, $\phi(x)$ is even.

**Fermat's Little Theorem**: If $x$ is a prime, $y^{x-1} \equiv 1 \mod x$, ($y$ is not divisible by $x$). Also, for any integer $y$, $y^x \equiv y \mod x$.

**Euler's Theorem**: Euler's Theorem states that $y^{\phi(x)} \equiv 1 \mod x$, for coprime integers $x$ and $y$.

This theorem can be used to prove the validity of $C = M^e \mod N$ and $M = C^d \mod N$ for $(M, C < N)$ if $e$ and $d$ are bound by the relation $ed = 1 + k\phi(N)$.

For $C = M^e \mod N$, $M$ can be calculated as:

$$\begin{aligned} M &= C^d \mod N \\ &= (M^e)^d \mod N \\ &= M^{1+k\phi(N)} \mod N \\ &= M.M^{k\phi(N)} \mod N \\ &= M \mod N \\ &= M \end{aligned}$$

The last equation is true for $(M < N)$. This method is used to prove RSA function.

### 2.1.4 Primality Testing

As large prime numbers cannot be generated by Mersenne and Fermat, there is a need to define an efficient method which can generate a large prime number. There are many primality testing methods which are deterministic or probabilistic. Deterministic methods always give the correct answer, but probabilistic methods may sometimes give the wrong answer. Due to the high complexity of deterministic methods, probabilistic methods came into existence. There are many probabilistic methods which are designed for primality testing. Probabilistic methods are efficient in terms of complexity, but these don't give the correct answer all the time. The possibility of the error is very less in these methods. For further reducing the error, the method must be run many times with different parameters. Following are some of the methods for primality testing. The first two methods are deterministic and remaining others are probabilistic methods:

**Trial Division Method**: This is the basic deterministic method which checks for the primality of a large random integer. By this method, the number $x$ is divided by all the primes less than $\sqrt{x}$, if any operation passes the divisibility test then $x$ is a composite number otherwise $x$ is sure to be prime. The complexity of this algorithm is $O(2^{n_x})$, where $n_x$ is the number of bits in $x$. This complexity is assumed to be exponential and this computation becomes non feasible once $x$ becomes large.

**AKS Method**: Another deterministic algorithm, AKS, designed in 2002, gave the initial complexity of $\mathrm{O}((\log n_x)^{12})$. The integer $x$ is prime if $((Z-k)^x \equiv (Z^x - k)$

mod $x$) is true for all integers $k$ coprime to $x$. The series is expanded in powers of variable Z and coefficients are compared.

**Fermat Test**: Integer $x$ is declared to be prime if $b^{x-1} \equiv 1 \mod x$, (here b is the base). But sometimes with probability $\epsilon$ it gives the same results for composite numbers. The composite numbers which pass this test are called Carmichael number. This probability of getting the correct prime number can be increased by computation with many bases. The complexity of this method is $O(n_x)$.

**Square Root Test**: In this method, all the numbers $y$ less than $x$ are squared and checked for modulo $x$. If $x$ is prime, then $\sqrt{y} \mod x = \pm 1$ for $y = 1$ only. For composite the result satisfies for other values of $y$ also. The result for composite may sometimes fail, e.g. for $x = 22$, only $\sqrt{1} \mod x = \pm 1$ is true.

**Miller Rabin Test**: This method gives the answer to whether the given number is composite, wherein 'yes' means the number is composite. Miller Rabin testing method is based on Fermat's Theorem and uses the fact that any odd integer $x$ can be represented as; $x - 1 = 2^m y$, here $(x - 1)$ is an even number, $m > 0$ and $y$ is any odd integer. The algorithm calculates $b^y \mod x$; (here, $b$ is a random number; $1 < b < x - 1$); if it comes out to be 1, the number might be a prime; otherwise square of $b^y \mod x$ is to be calculated if it comes out to be $-1$; the number might be prime; else this squaring step is repeated $m - 1$ times, at any step if the result is $-1$ the number might be a prime. If the result is never $-1$, the number is surely composite. With this test $x = 29$ comes out to be prime; which is correct and 221 comes out to be prime for a few random numbers, $b$; which is false. For reducing the error probability in answering as the prime number, this test is done for other positive random numbers (less than $x - 1$) also. The complexity of Miller Rabin test is $O(kn_x^3)$, where $n_x$ is the number of bits in the integer to be tested and $k$ is the number of different values of $b$ to be taken. By using efficient multiplication methods, complexity can be reduced to $O(kn_x^2)$.

By this study it is clear that the complexity of deterministic methods is very high, with probabilistic method one can find $O(n^3)$ as the best complexity.

### 2.1.5 Factorization Methods

There are two types of factorization method; general purpose and special purpose. General purpose factorization methods are dependent upon the length of

the number to be factored whereas special purpose factorization methods depend on the length of the smallest factor of the number to be factored. There are many factorization methods available in literature; a few are discussed here:

**Trial Division Method**: In this method the number $x$ is divided by all the numbers starting from 2 to $\sqrt{x}$, if any number divides $x$ then that number is a factor of $x$; this process has to be repeated until one finds all the factors. The complexity of this method is exponential in the number of bits in $x$.

**Fermat's Method**: This method is based on the fact that $x = a^2 - b^2 = (a + b) \times (a - b)$. The calculation of $a^2 - x$ (starting from $a = \sqrt{x} + 1$ and till $a < x$) is done until one gets the perfect square. The complexity of this method is sub exponential in the number of bits in $x$.

**Pollard p-1 Method**: This method is used to find the prime factor of the number $x$ and is based on the condition that the factor of $p - 1$ is not greater than a predefined value $a$. The method calculates $p = GCD(2^{a!} - 1, x)$, here $p$ is the prime factor; if $p > x$, then $x$ has no factor. The chance of failure is large if $a$ is not close to $\sqrt{x}$. The complexity of the method is exponential in the number of bits in $a$.

**Pollard rho Method**: The method is based on a function (say $f(z) = (z^2 + 1) \mod x$) and random number (seed) $z_1$. For factorizing the number $x$, the function $f(z)$ is calculated with this seed to have the value for $z_2$. If $GCD((f(z_1) - f(z_2)), x)$ is not 1, then the result is the prime factor of the composite number $x$, otherwise the process is repeated again with $z_i$ and $z_{2i}$, (i=2,3...) until one gets the $GCD((f(z_i) - f(z_{2i}), x) = p \neq 1$; $p$ is the required prime factor.The method fails to give answer if $p = x$. The prime number is expected to be smaller than $\sqrt{x}$ and the computation of this method requires $\sqrt{p}$ computations; thus the complexity of this method is $O(2^{n_x/4})$.

**Continued Fraction Factorization Method**: This method is based on prime factorization, in which continued fraction of $\sqrt{xN}$ is used ($N$ is the number to be factored and $x$ is the smallest value for which $xN$ is a square. The time complexity of this method is $O(e^{\sqrt{2 \log N \log \log N}})$.

**Quadratic Sieve Method**: This method is very efficient for the numbers less than 100 decimal digits and its complexity depends on the number to be factored

$(x)$. This method uses a sieving procedure to solve for the value of $z^2 \mod x$ with complexity $O(e^{\sqrt{\ln x \ln \ln x}})$.

**Number Field Sieve Method(NFS)**: This method uses a sieving procedure to solve for the value of $y^2 \equiv z^2 \mod x$ with the complexity $O(e^{2(\sqrt[3]{\ln x})(\sqrt[3/2]{\ln \ln x})})$.

**Elliptic Curve Factorization(ECM)**: This is considered as a special purpose factoring method as it is used to find the small factors. It uses elliptic curves to find the small factors. Its complexity depends upon the smallest factor of the composite number; $O(e^{(\sqrt{2}+o(1))(\sqrt{\ln p \ln \ln p})})$; here $p$ is the smallest factor of the number to be factored.

After the study of different methods it is clear that the complexity of general purpose methods is exponential or sub exponential in the size of the number to be factored. NFS can be used to factor small number; 232 digit number is the largest factored number till 2009 [11]. On the other hand, complexity of special purpose methods is exponential or sub exponential in the size of the smallest factor of the number to be factored. ECM is the best example for this type of numbers, 83 digit number is the largest factor computed in 2013 by this method[11].

### 2.1.6 Chinese Remainder Theorem

This theorem has its origin in an ancient Chinese puzzle. One can find out a number, if this number is divided by different prime numbers and all the remainders are known. This theorem is used to solve the congruent equations to give the unique solution if the equations are having different moduli relatively prime to each other.

The equations can be $M \equiv M_1 (\mod p_1)$, $M \equiv M_2 (\mod p_2)$, and so on up to $k$ equations. For solving the equations for the unique value of $M$, calculate $N = p_1 \times p_2 \times ... \times p_k$. Next find $N_1 = N/p_1$, $N_2 = N/p_2$, ..., $N_k = N/p_k$ and multiplicative inverse $(N_1)^{-1}$, $(N_2)^{-1}$,..., $(N_k)^{-1}$ with the given moduli $(p_1, p_2, ..., p_k)$. Subsequently $M$ can be found by calculating $((M_1 \times N_1 \times (N_1)^{-1} + M_2 \times N_2 \times (N_2)^{-1} + ... + M_k \times N_k \times (N_k)^{-1}) \mod N)$.

Garner's algorithm is used for the implementation of Chinese Remainder Theorem. Here the calculations are shown for modulus $N$ with two prime and the modulus $N$ with three primes.

For the modulus $N = p_1 \times p_2$; the steps are shown to calculate the value of $M$, given the value of $M_1, M_2, p_1\_inv\_p_2$, (here $p_1\_inv\_p_2 = (p_1)^{-1} \mod p_2$), $M$ can be calculated using Garner's algorithm:

$$M_{11} = M_2 - M_1 \mod p_2 \tag{2.1}$$
$$M_{11} = M_{11} \times (p_1\_inv\_p_2) \mod p_2 \tag{2.2}$$
$$M = M_{11} \times p_1 \mod N \tag{2.3}$$
$$M = M + M_1 \mod N \tag{2.4}$$

For three prime modulus $N = p_1 \times p_2 \times p_3$, given $M_1$, $M_2$, $M_3$, $p_1\_inv\_p_2$, $p_1p_2\_inv\_p_3$, (here $p_1\_inv\_p_2 = (p_1)^{-1} \mod p_2$, $p_1p_2\_inv\_p_3 = (p_1 \times p_2)^{-1} \mod p_3$), the following steps are computed by the Garner's algorithm [12]:

$$M_{11} = M_2 - M_1 \mod p_2 \tag{2.5}$$
$$M_{11} = M_{11} \times (p_1\_inv\_p_2) \mod p_2 \tag{2.6}$$
$$M_{12} = M_{11} \times p_1 \mod (p_1 \times p_2) \tag{2.7}$$
$$M_{12} = M_{12} + M_1 \mod (p_1 \times p2) \tag{2.8}$$
$$M_{11} = M_3 - M_{12} \mod p_3 \tag{2.9}$$
$$M_{11} = M_{11} \times (p_1p_2\_inv\_p_3) \mod p_3 \tag{2.10}$$
$$M = M_{11} \times p_1 \mod N \tag{2.11}$$
$$M = M \times p_2 \mod N \tag{2.12}$$
$$M = M_{12} + M \mod N \tag{2.13}$$

Both the Garner's methods, as shown above, are used in the implementation of the variants of RSA.

### 2.1.7 Continued Fractions

This is the special notation for the representation of any fraction $X/Y$ (here $X$ and $Y$ are not fractions). $X/Y$ can be represented as continued fractions:

$$\frac{X}{Y} = q_1 + \frac{1}{\frac{Y}{r_1}} \tag{2.14}$$

$$= q_1 + \frac{1}{q_2 + \frac{1}{\frac{r_1}{r_2}}} \tag{2.15}$$

$$= q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \frac{1}{\frac{r_2}{r_3}}}} \tag{2.16}$$

$$= q_1 + \frac{1}{q_2 + \frac{1}{q_3 + \frac{1}{q_4 + \ldots}}} \tag{2.17}$$

or

$$\frac{X}{Y} = q_1 + \frac{1}{q_2 +}\frac{1}{q_3 +}\frac{1}{q_4 + \ldots} \tag{2.18}$$

or

$$\frac{X}{Y} = [q_1; q_2; q_3; q_4; \ldots] \tag{2.19}$$

(here $q_i$'s are the quotients and $r_i$'s are the remainder). The process will be continued until numerator or denominator $\frac{r_x}{r_y}$ becomes 1. The expression for continued fraction can contain the finite terms or infinite terms which are called finite continued fractions or infinite continued fractions respectively. All $q_i$ and $r_i$ can be a whole number, real number, rational number or complex number. The above mentioned expression is simple continued fraction. The $i^{th}$ convergent of the expression is the truncation of the expression for $i$ values. If the values repeat, the expression is said to be periodic. The expression is always finite for rational numbers and it is infinite for irrational numbers.

One of the properties of continued fraction given below is used to attack RSA: If $x, y, z, w$ are positive integers and $GCD(x, y) = GCD(z, w) = 1$ such that

$$\left|\frac{x}{y} - \frac{z}{w}\right| < \frac{1}{2w^2} \tag{2.20}$$

then $\frac{z}{w}$ is one of the convergents of the continued fraction expansion of $\frac{x}{y}$.

### 2.1.8 Lattice Theory

Lattice $L$ is defined by all integer linear combinations of the linearly independent vectors; $u_1, u_2, ..., u_n \in \mathbb{Z}_m$, here $m \geq n$.

$L = \{u \in \mathbb{Z}_m | u = \Sigma_{i=1}^{n} a_i u_i\}$ with $a_i \in \mathbb{Z}$. The lattice is full rank for $m = n$. The set $B = \{u_1, u_2, ..., u_n\}$ is the basis of lattice $L$.

Determinant: For lattice $L$, say $< u_1, u_2, ..., u_w >$ are vectors in $\mathbb{Z}_n$, then determinant of the lattice can be defined as follows:

$$det(L) = \prod_{i=1}^{w} ||u_i^*|| \tag{2.21}$$

where $u_i^*$ is the orthogonal vector. For triangular matrix the determinant is calculated by just multiplying the entries of the diagonal. Every lattice $L$ with dimension $n$ has the shortest vector $u$ that is defined by the following relation:

$$|u| \leq \sqrt{n} \det(L)^{\frac{1}{n}} \tag{2.22}$$

For two dimensional Lattice $L$, the shortest vector can be found by the Gauss reduction algorithm. For arbitrary dimension, LLL reduction algorithm is used to find the approximate shortest vector. The lattice can be reduced by the LLL algorithm to get $< b_1, b_2, ...b_n >$, for $i = 1, 2, .., n$:

$$||b_i|| \leq 2^{\frac{n(n-1)}{4(n-i+1)}} \det(L)^{\frac{1}{n-i+1}} \tag{2.23}$$

Coppersmith [13] gave a new direction in cryptanalysis of RSA [14] and its variants by lattice reduction method. Coppersmith solved the polynomial congruences using Lattice theory, then Howgrave-Graham [15] simplified his approach. Coppersmith method can be used to find small solutions to the polynomial $f(x)$ modulo a large composite integer $N$ with unknown factors. The method basically find the roots of polynomial modular equations using lattice reduction. Multivariate linear polynomials with $k$ variables can be solved by generating $k$ algebraically independent equations and then finding the roots of these equations. Algebraic independence is assumed in almost all the cases; but proving the independence of polynomial is still an open problem. Thus this method is only heuristic. The following theorem based on the lattice theory is used in the work presented in the thesis:

For linear polynomial $f(x_1, x_2, ..., x_n)$ with integer coefficients, let $X_1, X_2, ..., X_n$ be the upper bounds for the roots $(x_{01}, x_{02}, ..., x_{0n})$ respectively.

If $M = ||f(x_1X_1, x_2X_2, ..., x_nX_n)||$, the roots $(x_{01}, x_{02}, ..., x_{0n}) \in \mathbb{Z}_n$ of $f$ can be calculated for $x_{01} < X_1, x_{02} < X_2, ..., x_{0n} < X_n$ if

$$X_1X_2...X_n < M \tag{2.24}$$

for sufficiently large $M$.

Also, for large composite integer $N$ with unknown factors; roots $(x_{01}, x_{02}, ..., x_{0n}) \in \mathbb{Z}_n$ of $(f \mod N)$ can be found if

$$X_1X_2...X_n < N \tag{2.25}$$

These bounds (2.24 and 2.25) are used in the cryptanalysis of RSA cryptosystem.

## 2.2 Standard RSA

RSA algorithm [4] was presented by Ron Rivest, Adi Shamir and Leonard Adleman in 1977. In RSA cryptosystem, pair $(e, N)$ is the public key and $d$ is the secret key. If sender $A$ wishes to send the message $M$ to $B$, $A$ uses the public key of $B$. If the message $M$ (a positive integer) is greater than $N$ (modulus), then $M$ is divided into small blocks of size less than $N$. The message $M$ is padded by standard method [16].

For the key generation method, two primes $p$ and $q$ of $n/2$ bits are chosen randomly, resulting in $n$ bit modulus $N = p \times q$ and $\phi(N) = (p-1) \times (q-1)$; here $\phi(N)$ is the Euler's totient function with about $n$ bits. Public exponent ($e$ with size $n_e$ bits) is chosen $(1 < e < \phi(N))$ such that $GCD(e, \phi(N)) = 1$. Private exponent ($d$ with size $n_d$ bits) is computed as the inverse of $e$ modulo $\phi(N)$. From the computation of the exponents $e$ and $d$, it is clear that only one of the exponents can be small, the other is calculated to be of the order of $\phi(N)$. The primes must be chosen as balanced prime numbers. Here balanced means,

$$4 < \frac{1}{2}\sqrt{N} < p < \sqrt{N} < q < 2\sqrt{N} \tag{2.26}$$

or $q \in \{p, 2p\}$

Instead of $\phi(N)$, $\lambda(N)$ is the necessary condition for RSA computations; $\lambda(N)$ is the Carmichael's lambda function; $\lambda(N) = LCM(p-1, q-1)$ and $\phi(N) = GCD(p-1, q-1) \times \lambda(N)$. Hence $\phi(N)$ is the multiple of $\lambda(N)$. With random prime generation, modulus N and $\phi(N)$ are approximately of the same size; also $\phi(N)$ and $\lambda(N)$ are roughly of the same size with high probability, as the $GCD(p-1, q-1)$ is assumed to be very small (say 2).

From the relation $ed \equiv 1 \mod \phi(N)$, RSA key equation can be written as:

$$ed = 1 + k_0\phi(N) \tag{2.27}$$

with $k$ as the security parameter. From the relation $k_0 = (ed-1)/\phi(N)$, $k_0$ is calculated to be $\min(e, d)$, as at least one parameter is of the order of $\phi(N)$.

For encrypting the message $M$ to get the cipher text $C$, message should be less than the modulus $N$ and relatively prime to it. If $GCD(M, N) > 1$, $GCD(C, N)$ will give the information about multiple of one of the prime factors. Cipher text $C$ can be found by computing the exponentiation of the message by the public exponent $e$; i.e.

$$C = M^e \mod N \tag{2.28}$$

The decryption can be done by:

$$M = C^d \mod N \tag{2.29}$$

The correctness of this method is based on Euler's theorem.

RSA security is dependent upon RSA problem. RSA problem is computing the $e^{th}$ root modulo a large composite integer. If the modulus $N$ is very large and $p$, $q$ and message $M$ are chosen randomly, RSA problem is assumed to be hard. One can recover the private exponent once the prime factors are known.

### 2.2.1 Efficiency of RSA operations

Complexity of RSA key generation is largely dependent on the prime generation. Random prime number is generated with complexity $O(n_p)$. The complexity of primality testing is $O(n_p^3)$, here $n_p$ is the number of bits in the prime factors of

the modulus. For the key generation method, random number is generated and tested for primality.

Complexity of encryption and decryption operation depends upon the exponentiation operation. This operation can best be implemented by square and multiply method. In this method, modular square operation is done for every bit (1 or 0) in the exponent and modular multiply operation is done for only 1 bit in the exponent. Both the operations (multiply or square) are performed with complexity $n^2$ [12], $n$ is the number of bits in the modulus. For $n_{e_x}$ multiply or square operations complexity becomes $n_{e_x}n^2$.

**Encryption Complexity:** In RSA encryption, for small public exponent $e$ (say $2^{16}+1$), only two bits are 1. Thus, 16 squares and 1 muliply modulo operations are performed. For $n$ bit modulus, the encryption complexity of RSA cryptosystem is: $Enc(C) = (16+1)n^2$, or

$$Enc(C) = 17n^2 \tag{2.30}$$

**Decryption Complexity:** In decryption method, private exponent $d$ is computed to be of the order of $N$. Here the number of 1's in $d$ (with size $n_d$ bits) are considered to be $n/2$ (for random primes). Thus $n$ squares and $n/2$ multiply modular operations are performed for the decryption complexity. For $n$ bit modulus, the decryption complexity of RSA cryptosystem is:

$Dec(C) = (n_d + n_d/2)n^2 = (n + n/2)n^2$

Thus

$$Dec(C) = (3/2)n^3 \tag{2.31}$$

## 2.2.2 Attacks on RSA Cryptosystem

### 2.2.2.1 Elementary Attacks

In this section, initial basic attacks are discussed:

**Cycling Attack**: In 1977, Simmons and Norris [17] showed that if the cipher text is encrypted repeatedly, one can get the cipher text again (say after $a+1$ encryption), the text found before this cipher text is the actual message (after $a$ encryption). For small messages, $a$ is very small. In 1979, Williams and Schmid [18] generalized the cycling attack. Friedlander, Pomerance and Shparlinski [19]

in 2001 showed that for balanced primes and sufficiently large modulus, cycling attack becomes ineffective.

**Common Modulus attack**: This attack is applicable in scenarios where the central authority distributes the private and public exponents to the users. All users share common modulus not having the knowledge of the prime factors. So if $d$ has somehow been leaked, the only work is to generate another private key from same $N$ and prime factors. In 1983, Simmons [20] showed that the message can be recovered if the same plaintext is encrypted with two different and relatively prime public exponents with the same modulus. In 1984, Delaurentis [21] showed the failure of common modulus RSA deterministically. He proved that the user can recover other's private key in the group with the use of his pair of public and private key. This attack doesn't work for a single user having multiple instances with the same modulus. In 1999, Howgrave-Graham and Seifert [22] described the strongest common modulus attack (single user) with a small private exponent. For two instances of $e$ and $d$ with same modulus, the security can be broken if $d < N^{\frac{5}{14}}$. For more than 7 instances with the same modulus the bound becomes, $d < N^{\frac{1}{2}}$ [23].

**Hastad Broadcast attack**: Common plaintext attack given by Hastad [24] in 1985, is for the same message encrypted with same public exponents to many users ($> e$) with different moduli. Hastad [24], [25] proved that the message $M$ can be computed in polynomial time. In 1997, Bleichenbacher [26] generalized this attack. He proved that with small public exponents and different moduli, if the messages are related to each other by some known function $f(x)$ and the number of users are greater than $max(e_i * degree(f_i(x))$, plaintext $M$ can be computed in polynomial time.

#### 2.2.2.2 Small-$e$ Attack

Small public exponent (say, $3, 5, 2^{16} + 1$) is always attractive for the low cost involved in the cryptosystem. But the low cost comes with its own risks.

If the message length is less than $N^{1/e}$, simply calculating the $e^{th}$ root of the cipher text will give the original message. This message fails when $M > N^{1/e}$, but if part of the message is known and the unknown part (contiguous or not) is less than $N^{1/e}$ then also the complete plaintext can be recovered by Coppersmith [27].

However, this attack fails for $e > \log(N)$, even for $e = 2^{16} + 1$ and also when at least $1/e$ part of the message is random.

In 1995, Franklin and Reiter [28] showed that if two plaintexts related with some function (with known constants) are encrypted by same small public exponent (e.g. $e = 3$), then the plaintexts can be computed in polynomial time. Coppersmith et al. [29] in 1996, showed that any number of plaintexts with a given relationship, encrypted with same public exponent, can be found in polynomial time. In 1997, Coppersmith [27] gave the random padding attack. He showed that two related plaintexts encrypted with a small public exponent ($e = 3$) can be computed even if the constants involved in the relationship are not known; i.e. for $M_2 = M_1 + c$, if $c$ is unknown and $|c| < N^{1/9}$, $M_1$, $M_2$ can be computed in polynomial time.

In 1998, Boneh, Durfee and Frankel [30] proved that for small $e$, $k_0$ in the RSA equation can be known (since $k_0 < e$). This gives the information about the MSBs (Most Significant Bits) of the private exponent; few bits (say $d_1$) can be known where $d - d_1 < p + q$. It means one can get half of the MSBs of $d$ for balanced primes. For $e = 3$, it is very simple to guess the value of $k_0$ (i.e. 2). Also, if one knows the bits of $d$, $k_0$ can be recovered easily. This attack is not the total failure of the cryptosystem. This attack signifies that for small $e$, half of the MSB of $d$ can be made public.

From this discussion, it is clear that public exponent $2^{16} + 1$ is assumed to be secure for RSA. Small public exponent leaks half of the MSBs of $d$ that is not the complete break of the system.

#### 2.2.2.3 Small-$d$ Attack

As the public and private exponents are inverse of each other, one can get small private exponent by selecting the large public exponent. If someone requires the fast decryption speed, small secret exponent is required. But very small secret exponent can break the complete system. The discussion for small $d$ is as follows:

In 1990, Wiener [31] gave the attack for balanced primes with small private exponent. If $d < \frac{1}{3}N^{\frac{1}{4}}$, one can factor $N$ in polynomial time. This attack can be completely avoided for $e > N^{\frac{3}{2}}$. Public exponent can be increased by adding the value of $\phi(N)$ to $e$. In 1997, Verheul and Tilborg [32] extended the Wiener's bound using exhaustive search. In 2003, by lattice method, May [35] rigorously proved

that RSA modulus with balanced primes can be factored in polynomial time if $d < \frac{1}{3}N^{\frac{1}{4}}$. In 2004, Dujella [33], extended the work of Verheul and Tilborg [32] for Wiener's bound (without exhaustive search). In 2009, Dujella [34] proposed a variant of Wiener's attack which is having less time and space complexity as compared to the Wiener's method.

In 1999, Boneh and Durfee [36] proved that for balanced primes, $N$ can be factored (heuristic method) if $d < N^{0.292}$. Blomer and May [37] in 2004 gave the variation of this attack. The bound of $N^{0.292}$ can be increased by exhaustive search to $N^{0.3}$.

In 1999, Howgrave-Graham and Seifert [22] introduced the bound of $d < N^{\frac{5}{14}}$ for two instances with common public exponent and different modulus. The bound becomes, $d < N^{\frac{1}{2}}$ for 7 instances [23].

In 2004, Coron and May [38] showed that modulus $N$ can be factored deterministically in polynomial time if private exponent $d$ is known and $e, d < \phi(N)$.

In 2002, De Weger [39] showed that small private exponent attacks become more effective if the difference between the primes is very small; i.e. when $N^{\frac{1}{4}} < |p - q| < N^{\frac{1}{2}}$. In 2008, Maitra and Sarkar [40] showed that Wiener's attack [31] gets improved when $|2q-p| < N^{\frac{1}{2}}$ with primes having common MSBs (Most Significant Bits). With shared common LSBs (Least Significant Bits) of primes, Zhao and Qi [41] in 2007, Sun et al. [42], [43] in 2008 gave their analysis on improved bounds for private key exponent. In 2007, Hinek [44] proved that if a user has different moduli with same private exponent, then for 3 instances the bound on the secret exponent becomes $d < N^{0.318}$. Hinek also showed that all small private exponent attacks also work for small negative private exponents with the same bound.

In 2005, Chen, Ku and Yen [45] presented a lattice-based attack for large values of private exponent close to $\lambda(N)$. The modulus can be factored if $d$ satisfies $|\lambda - d| < N^{0.25}$ [45]. This attack was further generalized by Hinek [44].

In 2014, Nitaj and Douh [46] proved that the RSA modulus can be factored if the secret exponent is of the form $(Xd_1 + d_0)$ for known value of X and unknown small values of $d_1$ and $d_0$.

#### 2.2.2.4 Partial Key Exposure Attack

Partial key exposure means some of the bits of secret exponent can be exposed by some method, say by exhaustive search or side channel attack. But the complete key cannot be derived by these means. Some of the consequences of the exposure of partial key are discussed here, $(\alpha, \beta, \delta$ is the representation of the fraction of public exponent (e), private exponent(d) and the unknown private exponent (if some of the bits are known) respectively):

In 1999, Boneh, Durfee and Frankel [30] stated that if $n/4$ LSBs of $d$ is known then modulus can be factored for $\beta = 1$ and $\alpha \in (0, 1/4)$.

They also presented attack for known $(1 - \delta)$ MSBs of $d$. If $e$ is prime or a composite with known factors and $\alpha \in (1/4, 1/2)$, the modulus can be factored if $\delta < 1 - \alpha$. If the factors of $e$ are not known and $\alpha \in (0, 1/2)$ then the modulus can be factored if $\delta < \alpha$ or $\delta < 1/4$.

Blomer and May [47] in 2003 presented partial key exposure attack for known LSBs of $d$. For $\beta = 1, \alpha \in (0, 7/8)$ and known $(1 - \delta)n$ LSB of $d$, modulus can be factored if $\delta < \frac{5}{6} - \frac{1}{3}\sqrt{1 + 6\alpha} - \epsilon$. Also for $\alpha \in (1/2, (\sqrt{6}-1)/2)$ and known $(1-\delta)n$ MSBs of $d$, modulus can be factored if $\delta \leq 1/8(5 - 2\alpha - \sqrt{36\alpha^2 + 12\alpha - 15} - \epsilon$.

In 2005, Ernst et al. [48] presented another partial key exposure attack for known LSBs and MSBs. In 2012, Joye and Lepoint [49] and in 2013, Nitaj [50], also presented some of the attacks based on partial knowledge of the secret key.

Following are some of the attacks based on the known bits of the prime factors:

Rivest and Shamir [51] in 1986 proved that 2/3 MSBs or LSBs of one of the primes are sufficient to factor the modulus.

In 1997, Coppersmith [27] showed that RSA modulus with balanced primes can be factored in polynomial time if $n/4$ of the MSBs or LSBs of prime $p$ are known.

In 2000, De Weger [39] showed that the small $d$ attack can be extended if the difference between the primes is small. Generally the difference between random primes is $\sqrt{N}$. Fermat's factoring technique can be applied to the primes with very small difference (less than $\sqrt{N}$). For prime difference less than $N^{1/4}$, Fermat's method is very fast. For very large prime difference, elliptic curve factoring attack can be applicable. Wiener's [31] and Boneh-Durfee [36] attack can be improved

with a small prime difference. With small prime difference these bounds work even for $\sqrt{N}$.

Maitra and Sarkar [40] in 2008, showed the exhaustive search attack on prime factors. Only 33 bits of the prime must be known to attack the system with $d \leq N^{0.3}$ [40].

In 2008, Herrmann and May [52] stated that the modulus can be factored if 0.7 bits (LSB or MSB) of one of the primes are known.

### 2.2.3 Improvement in Standard RSA

When the security of resource constrained environment is discussed, computational resources and memory consumption are the major parts. Slow computations involved in RSA cryptosystem are the main drawback. Complexity of computing the exponentiation modulo a large composite integer is very high. As the public and private keys are inverse of each other, one side always has to compromise for the speed loss. Various RSA variants are designed to improve the speed of RSA encryption/decryption operations and memory usage. Key generation time is not considered as very critical issue, as the keys are generated less frequently as compared to encryption/decryption processes. Encryption/decryption time and memory usage always affect the performance of device on which the cryptosystem is being used.

Although RSA history is full of variants in various improvement factors, here only a few of them are studied which make the foundation of the work presented in this thesis. Different improvement factors in the performance of RSA cryptosystem are shown in figure (2.1). As standard RSA is based on fast encryption speed (small size of $e$) and slow decryption speed (large size of $d$), basic variants of RSA were designed on the basis of the improvement over decryption speed. Basic variants [53] include RSA CRT [54], Batch RSA [55], MultiPrime RSA [56], MultiPower RSA [57] and Rebalanced RSA [31]. Other category of the improvement is balancing of both the encryption and decyption speed; e.g. Rebalanced RSA CRT Schemes (RRCS) [58], Tunable Balanced RSA [59]. Another critical aspect for improving the performance of RSA cryptosystem is memory consumption. Important variants based on the improvement in memory consumption are Twin RSA [60] and Dual RSA [61]. Basic variants are described in the following section. RRCS and

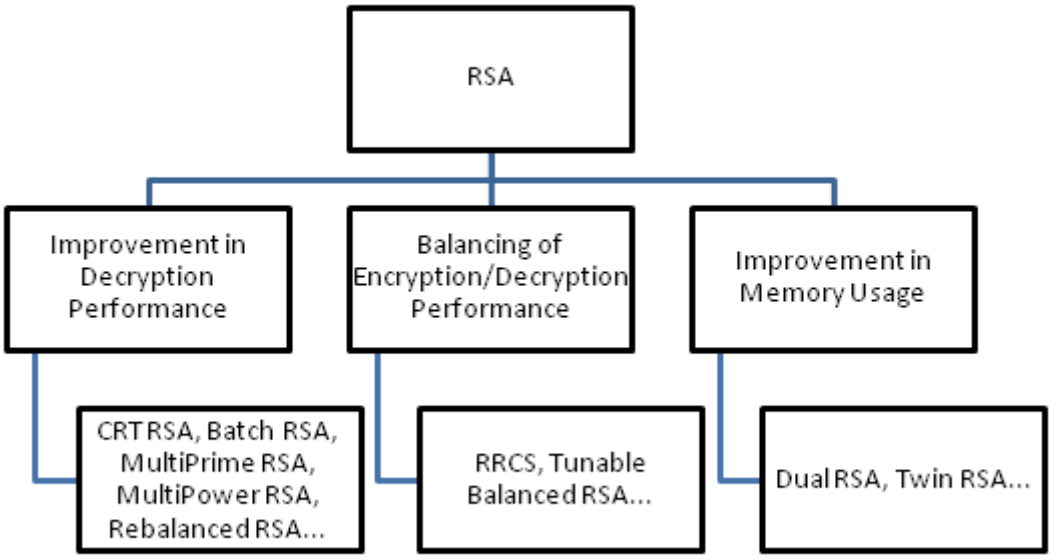


FIGURE 2.1: Classification of RSA Variants

Dual RSA are further elaborated in next sections as these two RSA variants are reworked in this thesis to achieve better performance.

## 2.3 Basic Variants (Improvement in Decryption Performance)

The most important part of RSA performance covers the decryption performance. Because of the security constraints, decryption exponent is usually considered to be very large of the order of the modulus size. Large size of the decryption exponent ($d$) lowers down the speed of decryption side. In this section, RSA variants which are based on the improvement in decryption speed are discussed. This is required in almost every case where RSA cryptosystem is used. Practical relevance of these variants lies in the signature generation (or decryption method) in heavily loaded web servers or small handheld devices, e.g. the bank customer is required to generate the signature (using his/her private key) on small device (like smart phones). The decryption method needs to be optimized in this case by reducing the computational complexity of the decryption method. This can be achieved by reducing the bit size of the decryption exponent in the computations involved in decryption method.

### 2.3.1 RSA CRT (Chinese Remainder Theorem)

In 1982, Quisquater and Couvreur [54] introduced this technique using CRT to improve the performance of decryption method in terms of computational speed so as to use RSA cryptosystem in small devices.

In RSA, the decryption exponent ($d$) is very large of the order of the modulus (n-bits), which makes the decryption process very slow. In RSA CRT, instead of working with large secret exponent ($d$), two smaller secret exponents, called CRT exponents ($d_p, d_q$), with size ($n/2$), bits are calculated. Cipher text is decrypted with these two smaller exponents resulting in two intermediate messages. Then these messages are combined with the help of CRT to obtain the original message.

Key generation and encryption methods are same as in standard RSA. For decryption method two private exponents ($d_p, d_q$) and two intermediate messages ($M_p, M_q$) are calculated with prime numbers $p$ and $q$:

$$d_p = d \mod p - 1 \tag{2.32}$$

$$d_q = d \mod q - 1 \tag{2.33}$$

$$M_p = C^{d_p} \mod p \tag{2.34}$$

$$M_q = C^{d_q} \mod q \tag{2.35}$$

Then by using CRT, $M$ can be computed by combining $M_p$ and $M_q$ (subsection 2.1.6)

Factor ($p^{-1} \mod q$) is calculated as part of the private key. If $d$ is less than the prime factors, the CRT exponents will be same as $d$. If CRT exponents are known, RSA CRT can be broken (also with the knowledge of one CRT exponent). RSA CRT will be broken by factoring the modulus. If $d$ is known then also the variant will fail. Like key equation in standard RSA (equation 2.27), RSA CRT is based on the following CRT key equations:

$$ed_p = 1 + k_p(p - 1) \tag{2.36}$$

$$ed_q = 1 + k_q(q - 1) \tag{2.37}$$

#### 2.3.1.1 Efficiency

Like standard RSA, two prime factors ($n_p = n/2$ bits) are used in the key generation method of RSA CRT. The complexity of random prime number is $O(n)$ and complexity of testing the primality is $O(n_p^3)$.

**Encryption Complexity**: In RSA CRT, public exponent can be taken to be small as in standard RSA. Thus the encryption complexity will remain the same; i.e.

$$Enc(C) = 17n^2 \tag{2.38}$$

**Decryption Complexity**: To calculate the decryption complexity, two small decryption exponents (or CRT exponents) are involved. Using square and multiply method, $n_d$ ($n/2$ bits) is the bit length of the exponent and $n_p$ ($n/2$ bits) is the bit length of the modulus involved in the computation. Square operations are $n_d$ and multiply operations are considered as $(1/2)n_d$. Computations are done for both the CRT exponents $(d_p, d_q)$. Thus, for two exponentiation operations (ignoring the computation for CRT); decryption complexity is,

$$\begin{aligned} Dec(C) &= 2 * (n_d + 1/2n_d)n_p^2 && (2.39)\\ &= 2 * (n/2 + 1/2 * n/2)(n/2)^2 && (2.40)\\ &= 1/4(3/2n^3) && (2.41) \end{aligned}$$

Hence the decryption speed is 4 times faster than standard RSA.

#### 2.3.1.2 Attacks

In RSA CRT, the encryption exponent ($e$) is taken to be small as described in standard RSA. Attacks defined in standard RSA with small $e$ are also applicable in this variant. As the private exponent ($d$) is taken to be of the size of modulus (n-bit), attacks based on small private exponent are not applicable in this case. Instead of the private exponent ($d$), two small CRT exponents $d_p$ and $d_q$ are taken. Thus in this variant attacks based on small CRT exponents are applicable. Here $e = N^{\alpha}$ and $d_p, d_q = N^{\delta}$ are considered wherever required; $\alpha$ and $\delta$ are the bit fractions of public and private exponents respectively.

**Small CRT Attacks**

RSA CRT is assumed to be free from small $d$ attacks because of the use of larger private exponent. The known method to attack small CRT exponents works in exponential time with complexity $O(min(\sqrt{d_p}, \sqrt{d_q}))$ [63].

In 2006 Bleichenbacher and May [62] gave cryptanalytic attack by which RSA primes for $q < N^{0.468}$ are insecure. They showed that the modulus can be factored in polynomial time if $d_p, d_q \leq min[(N/e)^{2/5}, N^{1/4}]$. This attack applies to Generalized RSA. Also, they described the attacks for $\delta < 7/12 - 1/12\sqrt{7 + 48\alpha} - \epsilon$ and $\delta < 2/5 - 2/5\alpha - \epsilon$.

RSA CRT is insecure if $\delta < 1/2 - 2/3\alpha - \epsilon$ and $\alpha > 1/4$ (for known $k_p$ and $k_q$) [59].

In 2006, Jochemsz and May [14] presented CRT attack, which states that if difference between the CRT exponents is known then the modulus can be factored in polynomial time for $\delta < 0.0986 - \epsilon$.

In 2007, Jochemsz and May [64] gave polynomial attack on small private CRT exponent. For full sized public exponent, modulus can be factored in polynomial time if $d_p$ (or $d_q$) is smaller than $N^{0.073}$ [64].

**Partial Key Exposure Attack**

In 2003, Blomer and May [47] proved that for balanced primes with known MSBs of $d_p$, modulus $N$ can be factored if $\beta < 1/4 - \alpha - \epsilon$ is unknown LSB fraction of $d_p$. Also the modulus can be factored for known LSBs of $d_p$ and known $k_p$ when $\beta < 1/4 - \epsilon$ is unknown MSB fraction of $d_p$. They added that for small $e$ (i.e. $e = 3$ or $2^{16} + 1$), half of the MSBs or LSBs of $d_p$ or $d_q$ are sufficient to factor the modulus.

### 2.3.2 Batch RSA

Fiat introduced Batch RSA [55] in 1989; the work was done to accomplish many decryption processes at the cost of approximately one. More than one jobs are combined to make a batch and decryption of the complete batch is performed in a single process, thus reducing the cost of multiple decryption processes.

This variant works for small and different public exponents for the same modulus N. Decryption of the two cipher texts in Batch RSA can be done at the cost of approximately one RSA decryption. Relevance of this variant is restricted to cipher texts with only very small public exponents and where decryptions have to be handled in bulk, e.g. in banks.

As this variant does not contribute much to the present work only the basic idea is given here. Concept of the computation can be understood by an example.

Key generation and Encryption methods are same as in standard RSA. Two messages ($M_1$ and $M_2$) are encrypted with small public exponents resulting in two cipher texts $C_1$ and $C_2$. Public keys for $C_1$ and $C_2$ are assumed to be $e_1 = 3$ and $e_2 = 5$ respectively.

For decryption process, $C_1$ and $C_2$ are calculated by computing the intermediate parameter X as:

$$X = (C_1^5 \times C_2^3)^{1/15} \tag{2.42}$$

$$C_1^{1/3} = X^{10}/[C_1^3 \times C_2^2] \tag{2.43}$$

$$C_2^{1/5} = X^6/[C_1^2.C_2] \tag{2.44}$$

(here $M_1 = C_1^{1/3}$ and $M_2 = C_2^{1/5}$)

Thus both the cipher texts $C_1$ and $C_2$ are decrypted to give $M_1$ and $M_2$ at the cost of single computations of $15^{th}$ root. Computations involved (calculation of $15^{th}$ root and some additional arithmetic) in the decryption method takes approximately same computational efforts as a single RSA decryption. Work on Batch RSA can be found in [65],[66],[67],[68], [69].

#### 2.3.2.1 Efficiency

Here the key generation method and encryption method is same as in standard RSA, so the complexity is same as in standard RSA.

As discussed above, the Decryption of b messages in Batch RSA is done with the cost of single exponentiation in standard RSA. Due to some extra computations involved, the practical speed up is not exactly b times. For $N = 1024$ bits,

decryption speed is increased by 2.6 with batch size 4 and by 3.5 with batch size 8 [55].

Any effective attack is not found in the study of Batch RSA. But to get the speed improvement there are some restrictions to be taken care of.

This technique is only valuable when the public exponents are distinct and small, say 3, 5, 7 etc., otherwise it will not increase the decryption speed; rather it will be more expensive. Also the modulus must be the same.

The main drawback of Batch RSA is that the server has to maintain different RSA certificates for every distinct key.

Batch size of more than eight is not suitable for common applications, as the delay will increase in the wait of the decryption requests.

Hence, the constraint of small public key exponent and delay introduced by waiting for the requests make this variant less attractive.

### 2.3.3 MultiPrime RSA

Collin et al. introduced this variant [56] in 1997. MultiPrime RSA was designed to enhance the decryption speed of RSA cryptosystem by taking more than two primes for the modulus. It consists of $k$ primes $p_1, p_2 ... p_k$ instead of using only two as in standard RSA. This variant is more suitable for use in resource constrained devices as it is more efficient in terms of computational speed as compared to RSA CRT.

For the key generation algorithm, with $n$ bit modulus($N$), $k$ distinct primes $p_1, p_2, ..., p_k$ of $n/k$ bits in length are chosen at random such that $N = \prod_{i=1}^{k} p_i$ and $\phi(N) \prod_{i=1}^{k}(p_i - 1)$. Public and private exponents ($e$ and $d$ respectively) are calculated in the same way as in standard RSA.

For balanced primes, MultiPrime RSA with $k$ primes satisfies the following relation:

$$4 < \frac{1}{2}N^{1/k} < p_1 < N^{1/k} < p_k < 2N^{1/k} \tag{2.45}$$

Encryption is done in the same way as in standard RSA. For decryption process, secret parameters $d_1$ to $d_k$ (CRT exponents) are computed as $d_i = d \mod (p_i - 1)$

and $M_1$ to $M_k$ are calculated as $M_i = C^{d_i}(\mod p_i)$. $M$ is calculated with $M_1$ to $M_k$ using CRT (subsection 2.1.6). Like RSA CRT key equations (equation 2.36, 2.37), MultiPrime RSA is based on $k$ CRT key equations ($k_i$ are the security parameter):

$$ed_i = 1 + k_i(p_i - 1) \tag{2.46}$$

MultiPrime RSA can be broken by factoring the modulus $N$. Once the private key $d$ or multiple of $\phi(N)$ is known, modulus can be factored probabilistically [70].

#### 2.3.3.1 Efficiency

In the key generation method, complexity of generating $n/k$ (for $k > 2$) bit prime is much less than generating $n/2$ bit primes in standard RSA and RSA CRT. In this variant keys are generated with small length prime factors ($n/k$-bits).

**Encryption Complexity**: Like standard RSA and RSA CRT, in MultiPrime RSA also the public exponent is taken to be small. Thus, the encryption complexity of MultiPrime RSA is,

$$Enc(C) = 17n^2 \tag{2.47}$$

**Decryption Complexity**: To calculate the decryption complexity, $k$ small decryption exponents are considered in this variant. Using square and multiply method, $n_d$ ($n/k$ bits) is the bit length of the exponent and $n_p$ ($n/k$ bits) is the bit length of the modulus involved in the computation. For all $k$ prime factors the decryption complexity is calculated with these small parameters and all the $k$ results are combined together to get the desired results. Thus, for $k$ exponentiation (ignoring the computation for combining the results); the decryption complexity is,

$$\begin{aligned} Dec(C) &= k * (n_d + 1/2n_d)n_p^2 && (2.48)\\ &= k * (n/k + n/2k)(n/k)^2 && (2.49)\\ &= 1/k^2(3/2n^3) && (2.50) \end{aligned}$$

the decryption speed is $k^2$ times faster than the standard RSA.

#### 2.3.3.2 Attacks

Attacks based on small public exponent described with standard RSA are also applicable with MultiPrime RSA. Lots of work is done in literature to analyse the effect of using multiple primes. This includes mainly the number of factors which can be used for the modulus. Here $e = N^{\alpha}$ and $d = N^{\delta}$ are considered wherever required; $\alpha$ and $\delta$ are the bit fractions of public and private exponents respectively.

**Factorization Attack**

The major problem with MultiPrime RSA is that one cannot increase the number of primes by choice because of the efficient factoring methods. There are many methods by which the composite modulus can be factored. NFS (Number Field Sieve) and ECM (Elliptic Curve Method) are the popular factorization methods. Complexity of NFS is $O(e^C)$, {here $C \approx 2(\ln N)^{1/3}(\ln \ln N)^{2/3}$}, thus the efficiency of this method is dependent upon the size of the modulus. NFS is general purpose factoring method as it is independent of the size and number of primes involved. On the other hand, the complexity of ECM is $O(e^C)$, { here $C = (2 \ln p \ln \ln p)^{1/2}$}, $p$ is the smallest factor of the modulus N. Hence, the efficiency of ECM is dependent upon the size of the smallest prime factor; it is special purpose factoring method. Thus to avoid the factoring attack, modulus must be large enough to resist general purpose factoring method and size of the primes must be large enough to resist the special purpose factoring methods. Analysis given in [71] shows implementation results of RSA with different modulus size and different number of prime factors. The time complexity of NFS increases with modulus size (1024 bits, 2048 bits, 4096 bits and so on). Efficiency remains the same with change in the number of primes. In case of ECM, complexity is shown to be decreased with increase in number of primes with same size of the modulus. The number of primes can be taken 3,3,4,5 as safe for modulus size 1024, 2048, 4096, 8192 bits respectively [71].

**Small d Attack**

In 2002, Hinek et al. [72] extended the Wiener's attack [31] for MultiPrime RSA. This attack describes that the private exponent can be computed in polynomial time if $d < N^{\frac{1}{2k}}/\sqrt{2(2k-1)}$.

In 2002, Ciet et al. [73] extended the Boneh and Durfee lattice attack [36] for MultiPrime RSA. The modulus can be factored probabilistically in polynomial time if $\delta \leq 1 - \sqrt{1 - 1/k} - \epsilon$ [73], (here $\epsilon$ is a very small value).

**Partial Key Exposure Attack**

If some of the LSBs or MSBs of all the primes but one (like half of the LSBs or MSBs of one out of two prime) are known, the modulus $N$ can be factored.

Boneh et al. [30] in 2001 presented partial key exposure attack for small public exponent. Private exponent can be computed for $N \equiv 3(\mod 4)$, $e \leq \frac{1}{8}N^{1/4}$ and known $d_0$ and $M$, where $d \equiv d_0 \mod M$ and $M \geq N^{1/4}$ [30].

Hinek et al. [72] presented attack for CRT exponents for choosing all but one CRT exponent as the small. Hinek et al. [72] in 2003 and [74] in 2008 extended many of the attacks proven on two prime RSA to $k$-prime RSA. The effect of every attack seems to be less on increasing the number of primes except the factoring attack. When some of the MSBs or LSBs of $(N - \phi(N))$ are known, the small private exponent attack and partial key exposure attack become more effective.

Multiprime RSA is insecure if $p_k - p_1 = N^{\alpha}$, $0 < \alpha \leq 1/k$, $k \geq 3$ and $2d^2 + 1 < n^{2/k-\alpha}/6k$ [75].

### 2.3.4 MultiPower RSA

Takagi introduced MultiPower RSA [57] in 1998. In this variant also, the purpose was to improve the decryption time of RSA algorithm. Instead of using multiple primes for the modulus, only two primes are used but with smaller sizes as compared to standard RSA. In the algorithm $N = p^{b-1}q$, where $p$ and $q$ are $n/b$ bits. Due to the use of only two primes MultiPower RSA is more efficient than MultiPrime RSA [56].

For the key generation algorithm, two primes $p$ and $q$ are chosen with $[n/b]$-bits each such that $N = p^{b-1} * q$. Like RSA, exponent $e$ is chosen randomly or with low hamming weight (less number of 1's in the bit representation), then $d$ is calculated such that $d = e^{-1} \mod \lambda(N)$, (here $\phi(N) = GCD(p-1, q-1) \times \lambda(N)$). CRT exponents are calculated as:

$$d_p = d \mod (p-1) \tag{2.51}$$

$$d_q = d \mod (q-1) \tag{2.52}$$

The encryption is done in the same way as in standard RSA.

For decryption,

$$M_p = C^{d_p} \mod p \tag{2.53}$$

$$M_q = C^{d_q} \mod q \tag{2.54}$$

thus $M_p^e = C \mod p$ and $M_q^e = C \mod q$

$M_p'$ is constructed such that

$$M_p'^e = C \mod p^{b-1} \tag{2.55}$$

Then $M$ is calculated using CRT such that $M = M_p' \mod p^{b-1}$ and $M = M_q \mod q$.

By knowing the multiple of $\lambda(N)$ or $d$ or one of the CRT exponent, Takagi's cryptosystem can be broken probabilistically by factoring the modulus.

#### 2.3.4.1 Efficiency

In the key generation method, only two primes are generated with $n/b$ bit size. Complexity of the random prime number is $O(n_p)$ and the complexity of primality testing $O(n_p^3)$ is used for generating two primes of length $n/b$; $(b > 2)$; which is less than standard RSA, RSA CRT and even MultiPrime RSA.

**Encryption Complexity**: Like previous variants, in MultiPower RSA also public exponent is taken to be small. Thus, the encryption complexity of MultiPower RSA is:

$$Enc(C) = 17n^2 \tag{2.56}$$

**Decryption Complexity**: To calculate the decryption complexity, two small decryption exponents are considered. Using square and multiply method, $n_d$ ($n/b$ bits) is the bit length of the exponent and $n_p$ ($n/b$ bits) is the bit length of the prime numbers involved in the computations. The square operations are $n_d$ and multiply operations are considered as $(1/2)n_d$. Computations are done with these small parameters. Thus, for two exponentiation (ignoring the computation for

combining the results); decryption complexity is,

$$\begin{aligned} Dec(C) &= 2 * (n_d + 1/2n_d)n_p^2 && (2.57)\\ &= 2 * (n/b + n/2b)(n/b)^2 && (2.58)\\ &= 2/b^3(3/2n^3) && (2.59) \end{aligned}$$

Hence, the decryption speed of Multipower is $b^3/2$ times faster than standard RSA.

#### 2.3.4.2 Attacks

Small public exponent attacks for standard RSA are also applicable for this variant. Other attacks are given below which include mainly the factorization attack due to the use of multiple factors. Here $e = N^{\alpha}$ and $d = N^{\delta}$ are considered wherever required; $\alpha$ and $\delta$ are the bit fractions of public and private exponents respectively.

**Factorization Attack**

As in MultiPrime RSA, in this variant also for $N = p^{b-1}q$, the value of $b$ is 3, 3, 4, 5 for modulus size 1024, 2048, 4096 and 8192 bits respectively to thwart the factoring attack [71] of ECM.

With another factoring attack known as Lattice Factoring Method (LFM) of Boneh et al. [76] in 1999, the modulus can be factored in polynomial time if at least $1/b$ of the MSBs or LSBs of $p$ or if at least $(b-1)/b$ of the MSBs or LSBs of $q$ are known. Also for large $b = \log p$, the LFM can factor the modulus in polynomial time. Even when $b = \sqrt{\log p}$, this method is faster than ECM factoring. In 2002, Chida et al. [77] gave another more efficient method based on lattice theory to factor the modulus with large $b$.

**Small d Attack**

In 2004, May [78] gave small private exponent attacks when $e$ and $d$ are calculated as modulo $\phi(N)$. If $\delta \leq (b-1)/b^2$, modulus $N$ can be factored in polynomial time. For $b \geq 3$, this attack becomes stronger.

In 2008, Itoh et al. [79] generalized the attack of Boneh and Durfee's lattice attack [36]. If exponents $e$ and $d$ are defined modulo $\phi(N)$, the modulus $N$ can be factored in polynomial time for $\delta < (2 - \sqrt{\alpha b})/b - \epsilon$. Also, it is shown that the private exponent $d$ can be recovered if $\delta < (2 - \sqrt{2})/(b+1)$.

The modulus can be factored if $\delta \leq ([b-2]/b)^2$ [80]. This attack is strongest for $b \geq 4$. Sarkar [81] in 2013 proved that this variant is insecure for $d < N^{0.395}$.

**Partial Key Exposure Attack**

May [80], [78] in 2004 presented some of the partial key exposure attacks on Multipower RSA. In 2013, Lu, Zhang, Lin [82] showed that if $[\ln(1+b)]/b$ MSBs or LSBs of $p$ are known, the modulus can be factored in polynomial time.

### 2.3.5 Rebalanced RSA

Rebalanced RSA [31] was introduced by Wiener in 1990. This variant increases the decryption speed at the cost of slow encryption speed. In this variant the decryption speed is enhanced by shifting most of the work to encryption side, resulting in high speed of decryption method and slow speed of encryption method. Some applications require this property, e.g. signature generation (equivalent to decryption) is required on small devices (like smart phones) and signature verification (equivalent to encryption) on large servers.

For the key generation method, the prime factors $p$ and $q$ of $n/2$ bits are selected at random such that $\text{GCD}(p-1, q-1) = 2$ and $N = pq$. Two CRT exponents $d_p$ and $d_q$ of $n_d$ bits are selected at random such that $GCD(d_p, p-1) = GCD(d_q, q-1) = 1$ and $d_p = d_q \mod 2$. By using CRT, $d$ is calculated such that $d = d_p \mod (p-1)$ and $d = d_q \mod (q-1)$. The public exponent $e$ is calculated by taking the inverse of $d$ modulo $\phi(N)$. Hence $e$ comes out to be of the order of $\phi(N)$. Encryption and decryption methods are same as in RSA CRT [54].

In 2003, this variant was combined with MultiPrime RSA, to give the improved variant RPrime (Rebalanced RSA and MultiPrime RSA) [83] in terms of speed at decryption side. For the key generation of RPrime RSA, $k$ primes are chosen at random and then the steps of Rebalanced RSA are followed to calculate $k$ CRT exponents $d_1$ to $d_k$. Encryption and decryption steps are same as in MultiPrime RSA.

#### 2.3.5.1 Efficiency

In the key generation method, two prime factors are considered with length $n/2$ bits ($n$ is the bit length of modulus N). Though the size of the parameters in the

complexity is same as in standard RSA, but here the complexity is more because of the condition to be satisfied for the random prime.

**Encryption Complexity**: Unlike standard RSA and other variants discussed above, the encryption exponent is not small due to the inverse of small decryption exponent. Encryption exponent is of the order of the modulus. By square and multiply method, $n$ squares and $n/2$ multiply operations are done with modulus $n$ bit size. Thus the encryption complexity is about $(n + n/2)n^2$; i.e.

$$Enc(C) = 1.5 * n^3 \tag{2.60}$$

**Decryption Complexity**: To calculate the decryption complexity, two small decryption exponents are considered in this variant. Using square and multiply method, $n_d$ (say 160 bits) is the bit length of the exponent and $n_p$ ($n/2$ bits) is the bit length of the modulus involved in the computation. The square operations are $n_d$ and multiply operations are considered as $(1/2)n_d$. Two decryption calculations are done with small parameters and the results are combined with CRT. The decryption complexity is,

$$\begin{aligned} Dec(C) &= 2 * (n_d + 1/2n_d)n_p^2 && (2.61)\\ &= 2 * (1.5 * 160)(n/2)^2 && (2.62)\\ &= 120n^2 && (2.63) \end{aligned}$$

For the above calculation, the decryption exponents (CRT exponents) are taken to be of size 160 bits. Hence, the complexity is much smaller than all the previously mentioned variants.

#### 2.3.5.2 Attacks

Attacks described for RSA CRT are also applicable for Rebalanced RSA. One has to take care that in RSA CRT the public exponent is small, but here it is large.

## 2.4 Balancing of Encryption/Decryption Performance

In some applications, there is a need to balance both the sides (encryption/decryption) of the cryptosystem; e.g. when two small devices are communicating with each other, encryption or decryption performances should not be biased. Practical relevance of balancing of the encryption and decryption methods can be found in smart phone to smart phone communication.

Two independent variants were proposed in 2005 based on balancing of encryption and decryption methods. One is Rebalanced RSA CRT Schemes (RRCS) by Sun and Wu [84] to decrease the encryption time of Rebalanced RSA CRT. Another is Tunable Balanced RSA [59] which was designed independent of RRCS resulting in balancing of public and private operations. Difference between Tunable balancing RSA and Rebalanced RSA CRT variants is that the former is more general than the later. The work in this thesis is dependent on RRCS, this variant is elaborated below.

### 2.4.1 Rebalanced RSA CRT Scheme (RRCS)

RRCS [58] is the improvement over Rebalanced RSA, so that the increased encryption cost can be lowered to balance both encryption and decryption cost. In this variant the user is given the choice of selecting the bit-size of public and private exponents as per the requirement. The exponents are so chosen as to balance the encryption and decryption computational performance or to bias any of the sides.

In RRCS[58] two schemes are described: RRCS($n_e < n/2$) and RRCS($n_e > n/2$).

In RRCS($n_e < n/2$), public exponent $e$ is chosen at random or with low hamming weight (less number of 1's in the bit representation) with size ($n_e < n/2$). The parameters $k_p, k_q, d_p, d_q$ can be calculated to satisfy the RSA CRT key equations, $ed_p = k_p(p-1)+1$ and $ed_q = k_q(q-1)+1$. The fast encryption speed of this variant is at the cost of a slight decrease in decryption speed.

In RRCS($n_e > n/2$), public exponent $e$ is chosen at random with size ($n_e > n/2$). In this scheme also, one is having the choice of selecting $e$ with low hamming weight to further reduce the encryption complexity. Other parameters $k_p, k_q, d_p, d_q$ can

be calculated such as to satisfy the RSA CRT key equation, $ed_p = k_p(p-1)+1$ and $ed_q = k_q(q-1)+1$. In this scheme, factoring method is used to factor a small number, which makes the key generation method complex.

In both the schemes, encryption and decryption methods are same as in RSA CRT [54].

#### 2.4.1.1 Efficiency

Like RSA CRT, this algorithm generates two small decryption exponent with two primes. But the constraints of generating these parameters are more than the constraints of RSA CRT. The complexity of prime number generation is $O(n_p)$ and primality testing is $O(n_p^3)$. In case of RRCS($n_e > n/2$), the complexity further increases because of the use of factoring method.

**Encryption Complexity**: For RRCS($n_e < n/2$), 170-bit public exponent is used ($e = 2^{169}+1$). Hence for square and multiply method 169 squares and 1 multiply operations are done with modulus size n/2. The encryption complexity is:

$$Enc(C) = 170n^2 \tag{2.64}$$

For RRCS($n_e > n/2$), 592-bit public exponent is used ($e = 2^{591}+1$). Hence for square and multiply method 591 squares and 1 multiply operations are done with modulus size n/2. The encryption complexity is:

$$Enc(C) = 592n^2 \tag{2.65}$$

**Decryption Complexity**: For decryption method in RRCS($n_e < n/2$), two 358-bit CRT-exponents $d_p, d_q$ are used. For square and multiply method, 358 squares and 358/2 multiply operations are performed for both the CRT exponents. Thus the decryption complexity is:

$$\begin{aligned} Dec(C) &= 2 * (358 + 358/2)(n/2)^2 && (2.66)\\ &= 268n^2 && (2.67) \end{aligned}$$

For RRCS($n_e > n/2$), two 190-bit CRT-exponents $d_p, d_q$ are used. For square and multiply method, 190 squares and (190/2) multiply operations are performed for both the CRT exponents. Thus the decryption complexity is: Dec(C)= $142n^2$

#### 2.4.1.2 Attacks

All the attacks presented for RSA CRT are also applicable to this variant. The scheme is proved to be secure against small $d$ attack by Wiener [31] and Boneh-Durfee [36]. It is also shown to resist lattice attack by Coppersmith [27]. Besides the attacks described with CRT variant, the wrong choice of parameters (size of $e$, $d_p$, $d_q$) makes the security of the variant weak. Thus, the security needs to be discussed according to the parameter selection. Here $e = N^{\alpha}$ and $d_p, d_q = N^{\delta}$ are considered wherever required; $\alpha$ and $\delta$ are the bit fractions of public and private (CRT) exponents respectively.

**Parameter Selection Attack**

In 2002, May [85] presented the attack to factor the modulus for balanced exponents if $\delta < 3/8 - 1/2\alpha - \epsilon$. For small $e$ and the small CRT exponent this attack is not effective.

Bleichenbacher and May [62], in 2006, stated that the modulus can be factored in polynomial time if $d_p, d_q \leq min\{(N/e)^{2/5}, N^{1/4}\}$.

Various lattice based attack can be mounted on the scheme if the following conditions are satisfied (here $\gamma = m/N$; $m$ bits are known by exhaustive search):

$$5\delta + 2\alpha < 2 + \gamma \tag{2.68}$$

$$3\delta + 2\alpha < 3/2 + \gamma_1 or \alpha > 1/4 - \gamma_2 (\gamma = \gamma_1 + \gamma_2) \tag{2.69}$$

$$6\delta + 3\alpha < 5/2 + \gamma \tag{2.70}$$

$$6\delta + 3\alpha < 7/3 - \gamma - (1/2)(4\gamma_2 - 4\gamma - 12\gamma\alpha + 6\alpha)^{1/2} \tag{2.71}$$

The scheme can be broken with relation $4\alpha + 14\delta < 5 - \epsilon$ [64] (for very small $\epsilon$).

If the security parameters ($k_p$ and $k_q$) are known then the modulus can be factored [59] if $n_e \geq n/4 - m$. If $k_p$ and $k_q$ are not known, then if $n_k \leq (n_e + m)/3$ these parameters can be calculated, (here $n_k$ is the bit size of $k_p$ and $k_q$). Lattice attack can be mounted on this variant if $n_d + 4n_k \leq 2n_e + 4m$.

## 2.5 Improvement in Memory Consumption

As RSA cryptosystem consumes large amount of memory, lots of research has been done in this area to reduce the memory requirement. In 1994, Vanstone and Zuccherato scheme [86] was presented which was based on the generation of RSA moduli with a predetermined set of bits. This scheme was attacked by Coppersmith [13]. After their scheme many improvements in memory consumption were considered in literature [87],[88],[60],[89],[90],[91],[92].

Here Dual RSA [61] is discussed which is based on the improvement in memory consumption by two instances of RSA. Dual RSA has a big impact on the work presented in this thesis. This variant has the relevance in low memory devices which are used to generate blind signatures or authenticity/secrecy.

### 2.5.1 Dual RSA

In 2007, Sun et al. [61] proposed Dual RSA to reduce the memory requirement. Dual RSA generates the same public and private exponent for two RSA instances. The public and private exponents $(e, d)$ are generated and shared by two instances with different modulus. Sharing of the parameters by two RSA instances reduces the memory requirement by the cryptosystem.

Three schemes are proposed for Dual RSA; Dual RSA Small-e, Dual RSA Small-d and Dual Generalized Rebalanced RSA (DGRR). These three schemes are based on small public exponent, small private exponent and balanced public/private exponents respectively. The schemes are suitable for applications like blind signatures and authentication/secrecy.

#### Dual RSA Small-e

This scheme is based on small public exponent for two instances. This method is based on two RSA key equation:

$$ed = 1 + k_1\phi(N_1) \tag{2.72}$$

$$ed = 1 + k_2\phi(N_2) \tag{2.73}$$

For the key generation method, $(p_1 - 1), (p_2 - 1)$ and $(q_1 - 1)$ are calculated by $x_1x_2, x_1y_2, y_1y_2$ respectively. Here $x_1, y_1$ are random numbers with size $n_e$-bit and $x_2, y_2$ are random numbers with size $(n/2 - n_e)$ bit. Public exponent $e$ is selected at random such that it is coprime to $x_1x_2y_1y_2$. Further the private exponent $d$ and the security parameter $k_1$ are computed with the RSA key equation $ed = 1 + k_1(p_1 - 1)(q_1 - 1)$. Prime $q_2$ is calculated by $(k_1x_2 + 1)$. Two RSA instances (moduli $N_1 = p_1q_1$, $N_2 = p_2q_2$) are generated with common public and private exponents ($e$, $d$) .

**Dual RSA Small-d**

This scheme is based on small private exponent for two RSA instances. Keys for this scheme are generated by the same method as RSA Small-e. The only difference is that the private exponent is chosen to be small and then public exponent is computed with the RSA key equation resulting in the size of the order of modulus.

**Dual generalized Rebalanced RSA(DGRR)**

This scheme is based on the Generalized RSA variant, where the public and private exponents can be selected on the basis of the requirement. This method is based on the following RSA key equation:

$$ed_p = 1 + k_{p_1}(p_1 - 1) \tag{2.74}$$

$$ed_p = 1 + k_{p_2}(p_2 - 1) \tag{2.75}$$

$$ed_q = 1 + k_{q_1}(q_1 - 1) \tag{2.76}$$

$$ed_q = 1 + k_{q_2}(q_2 - 1) \tag{2.77}$$

here $n_e < n/2$ and $n_e + n_d = n/2 + n_k$ (here $n_k$ is the bit size of the security parameter $k_i$). For the key generation method, public exponent $e$ is chosen randomly and $k$ is calculated as $k = [(n/2 - n_e)/n_k]$. Parameter $p_a$ is calculated as the product of $k$ small random numbers and random parameter $k_{p1}$ is selected such that $p_a$ and $k_{p1}$ are coprime to public exponent. Then $d_p$ and $p_b$ are calculated with the RSA CRT key equation. Prime factor $p_1$ can be calculated by $p_ap_b + 1$, and $p_2$ by $k_{p_1}p_ap_b/p_{a_i}$, where $p_{a_i}$ is among any of the factor of $p_a$. This process is repeated to calculate $q_1$ and $q_2$.

#### 2.5.1.1 Memory Usage

In Dual RSA, memory consumption is calculated for two RSA instances; public exponent ($e$) and private exponent ($d$) are shared by both the instances. Memory consumption by Dual RSA [61]:

Dual RSA Small-e:

$$Mem(C) = (3.25n + l/2) \tag{2.78}$$

Dual RSA Small-d:

$$Mem(C) = 3.333n \tag{2.79}$$

DGRR:

$$Mem(C) = 2.7n + l/5 \tag{2.80}$$

(where $l$ is the number of bits known by exhaustive search)

#### 2.5.1.2 Attacks

Attacks defined with the basic variants of RSA are not applicable directly for this variant. Dual RSA is prone to the attacks which are based on taking two instances. As Dual RSA is based on small $e$, small $d$ and generalized parameters; all these factors are considered here. Here $\alpha$ and $\delta$ are the bit fractions of public and private (CRT) exponents respectively.

**Small $e$ Attacks**

- The modulus can be factored if $\alpha < 1/4 + \epsilon$.
- Modulus can be factored for $\alpha < 1/4 + l/2 - \log_N 19$, (or $n_e < 332$).
- If security parameters ($k_1$ and $k_2$) are known, the modulus can be factored if $e$ is very small or very close to $n/2$ (i.e. $e < 80$ or $e > 432$).

**Small $d$ Attacks**

- Lattice based attack for Dual RSA small-e applies to this scheme also; modulus can be factored if $\alpha + \delta < 5/4 + l/(2 \log N) - \epsilon$, or $n_d < 296$

- Modulus can be factored if $k_1/k_2$ is known and $\alpha + \delta > 1 + \gamma - \epsilon$, for $\alpha \approx 1$, $\delta > \gamma - \epsilon$.
- Private exponent $d$ can be calculated if $\delta + 2/3\alpha < 1$ or $\delta < 1/3$.

**Parameter Selection Attacks**

- Modulus can be factored for the following constraints [61] (here $l$ is known):

$$5n_d + 2n_e > 2n + l \tag{2.81}$$

$$4\alpha + 14\delta < 5 - \epsilon \tag{2.82}$$

$$n_k < n/8 + l/2 - 1 (or) n_e + n_d > 5n/8 + l/2 - 1 \tag{2.83}$$

- The modulus can be factored if the CRT exponents satisfy the relation $\delta < 2l/\log N$ [44]. Also, the following bounds threaten the security of the scheme:

$$4\alpha + 6\delta < 3 \tag{2.84}$$

$$\delta < 1/2 - \epsilon \tag{2.85}$$

(The above bound is stronger for $\alpha < 3/8$)

$$\alpha + \delta < 5/8 - \log(8\sqrt{7})/2 \tag{2.86}$$

and

$$\alpha + \delta > 1/2 + \log(k') - \epsilon \tag{2.87}$$

(here $k' = GCD(k_1, k_2)$)

- Maitra and Sarkar [93] in 2013 proved that for $\alpha > 1/4$, Dual RSA modulus can be factored if $\delta < (1 - \alpha)/2 - \epsilon$. They also proved that the modulus can be factored for $\alpha > 1/4$, if $\delta < (3 - 4\alpha)/5$ and $\alpha + \delta > 1/2 + \epsilon$.

## 2.6 Comparison of RSA Variants

Different categories of RSA variants are dicussed above, all of these have their own relevance in different applications. Among all the variants presented above, RSA CRT, MultiPrime RSA, Rebalanced RSA, RRCS and Dual RSA are important in

this work. MultiPrime RSA (better than RSA CRT) can be used in the communication between small device (encryption) and server (decryption), Rebalanced RSA can be used in the communication between server (signature verfication/encryption) and small device (signature generation/decryption), Generalized RSA can be used in the communication between two small devices (where encryption and decryption performance can not be biased). Dual RSA is more application specific variant which is used in blind signatures and authenticity/secrecy. As these variants are found popular in RSA literature, in this work also improvement in the performance of RSA cryptosystem is made on the basis of these variants.

In this section, RSA variants discussed above are compared (Table 2.2 and figure (2.2)) on the basis of computational performance. Batch RSA and MultiPower RSA are not considered for comparison purpose as these two variants are not related to the work presented in this thesis. Batch RSA is special variant which is suitable in the scenario where more than one decryption is to be done by making a batch. In MultiPower RSA, large size encryption exponent (e) is not allowed.

In Table 2.2, the values are shown by assuming the size ($n$) of the modulus ($N$) as 1024 bits and number of primes is considered to be 3 in MultiPrime case.

TABLE 2.2: Comparison of Existing RSA variants based on Computational Performance

| Sr. No. | RSA Variants | Public Exponent ($n_e$ bits) | Encryption Complexity | Private Exponent ($n_d$ bits) | Decryption Complexity |
|---|---|---|---|---|---|
| 1. | RSA[4] | 17 | $17n^2$ | 1024 | $1536n^2$ |
| 2. | RSA CRT[54] | 17 | $17n^2$ | 512 | $384n^2$ |
| 3. | MultiPrime RSA[56] | 17 | $17n^2$ | 341 | $170n^2$ |
| 4. | Rebalanced RSA[31] | 1024 | $1536n^2$ | 160 | $120n^2$ |
| 5. | RRCS($n_e < n/2$) [58] | 170 | $170n^2$ | 358 | $268n^2$ |

Another factor discussed above in RSA performance is memory usage. Table 2.3 shows the comparison of different RSA variants based on the consumption of number of bits. Comparison is shown for two instances of RSA. Memory consumption is computed as the total bits consumed by modulus ($N$, $n$ bits), public exponent ($e$, $n_e$ bits) and decryption exponent ($d$, $n_d$ bits).

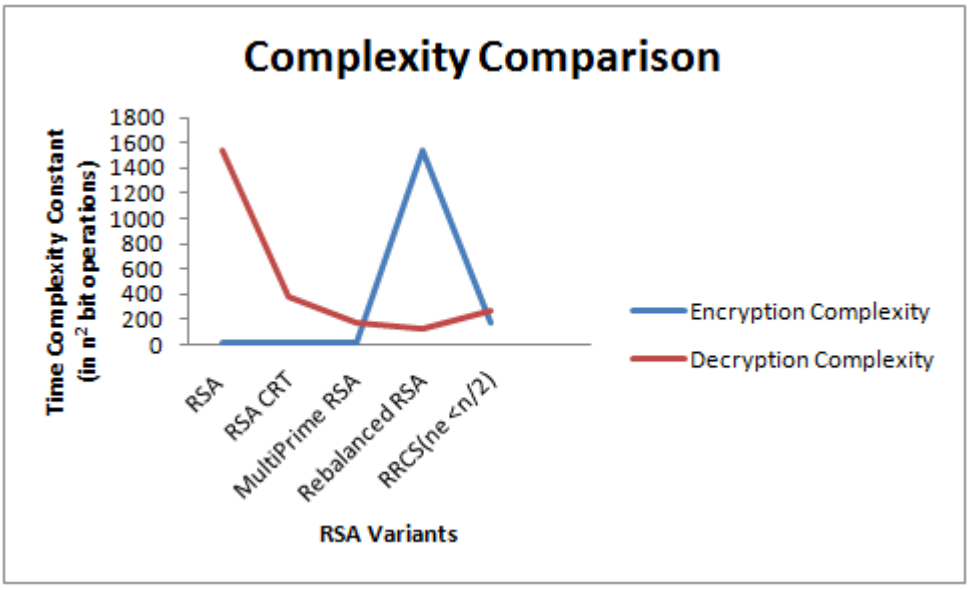


FIGURE 2.2: Comparison of existing RSA Variants based on Computational Performance

As standard RSA, RSA CRT and MultiPrime RSA consume same numer of bits, these variants are counted only once in the comparison. The memory consumption is calculted as $(n + n_e + n_d = 2n + 17 + 2n)$. For two instances, 2n bits for storing the two modulus, 17 bits for the encryption exponent and 2n bits for decryption exponent. Thus all these variants are categorized under the name of RSA.

TABLE 2.3: Comparison of existing RSA variants based on Memory Usage

| Sr. No. | RSA Variants | Modulus (N) (#bits) | Public Exponent (e) (#bits) | Private Exponent (d) (#bits) | Total Memory by two RSA instances (#bits) |
|---|---|---|---|---|---|
| 1. | RSA [4] | n | 17 | n | 4n+17 |
| 2. | Rebalanced RSA [31] | n | n | 2*160 | 4n+640 |
| 3. | RRCS ($n_e < n/2$) [58] | n | 170 | 2*358 | 2n+1772 |
| 4. | Dual RSA (Small e) [61] | n | n/4+m/2 | n | 3.25n+40 |

In case of balanced encryption/decryption exponent, modulus size $n$ is same and size of the encryption and decryption exponent ($n_e$ and $n_d$) varies according to the requirement, RRCS ($n_e < n/2$) (or any variant based on balancing of the parameters can be taken) is also considered here. As standard RSA is considered for small e, Dual RSA [61] is also taken for small e in the table (Table 2.3).

Besides the variants described above, there are many other improved variants in the literature of RSA cryptosystem. Unbalanced RSA [94], [95] was introduced by allowing one of the primes as small number. The scheme was broken in [96], [85], [62]. Common Prime RSA [97] was designed by Hinek in 2006 for reducing the decryption speed and memory consumption. Weak parameters for Common prime were shown in [98], [64], [31], [93]. Sarkar and Maitra [99] in 2010 introduced the variant with low hamming weight (less number of 1's in the bit representation). The variant was shown to be insecure by Sarkar and Maitra [100].

# Chapter 3

# Improvement in Rebalanced RSA CRT Scheme

After the study of RSA variants in chapter 2, it is found that some of the variants can be improved to exhibit better performance in terms of computational speed. Rebalanced RSA CRT Scheme (RRCS) is found to have balanced encryption and decryption speed. In this chapter, RRCS is further extended to design a new variant; Extended RRCS (ERRCS). ERRCS results in improved decryption speed as compared to RRCS while maintaining the same encryption speed. Various security attacks for the proposed scheme are discussed in section 3.2. Efficiency of the proposed scheme is analyzed theoretically and by implemention in section 3.3 and 3.4 respectively. In section 3.5, ERRCS is compared with other RSA variants and decryption speed of the proposed scheme is shown to be 1.9 times faster as compared to RRCS.

## 3.1 Extended Rebalanced RSA CRT Scheme (ERRCS)

For signature generation (decryption method) on small devices, Rebalanced RSA [31] was designed with fast decryption speed but slow encryption speed. The variant is found to be suitable for small devices (like smart phones) communicating with the devices with lots of computational resources (like servers). This variant is not suitable in scenarios where both the communication ends have limited

resources. To overcome this problem in such scenarios, Rebalanced RSA was improved to design Rebalanced RSA CRT Schemes (RRCS)[58]. In RRCS, work is done on Rebalanced RSA to increase the encryption speed such as to achieve the balance in encryption and decryption speed. Here RRCS is further extended as ERRCS for improvement in decryption speed.

RRCS was designed for small bit-size of public exponent ($n_e < n/2$) using two prime modulus ($N = p_1 * p_2$) satisfying the CRT key equations (eq. 2.36 and eq. 2.37) with bit size $n_e + n_d = n_k + n/2$. In the proposed scheme (ERRCS), modulus N is considered with multiple primes ($N = p_1 * p_2 * ...p_x$). Due to multiple prime factors ($x$), public and private exponents are calculated such as to satisfy CRT key equation (2.46) with bit size $n_e + n_d = n_k + n/x$. Small bit-size of the public and private exponents with $x$ primes improves the performance of the ERRCS.

### 3.1.1 Key Generation Method

For key generation method, $n$ is the number of bits in modulus (N), $n_e$ is the number of bits in public exponent (e), $n_d$ is the number of bits in CRT exponents ($d_{p_i}$) and here $n_e < n/x$ for $x$ primes). Algorithm for key generation method is as follows:

1. Select a random ($n_e$)-bit odd integer $e$.
2. Select ($n/x - n_e$) odd integer $d_{p_{1a}}$ and calculate $E_{p_1} = ed_{p_{1a}}$.
3. Select random number $k_{p_1}$ with size ($n_e + n_d - n/x$) such that $GCD(k_{p_1}, E_{p_1}) = 1$.
4. Calculate $d_{p_{1b}}$ and $p_{1a}$ such that $E_{p_1} d_{p_{1b}} = k_{p_1} p_{1a} + 1$, here $k_{p_1} < d_{p_{1b}} < 2k_{p_1}$ and $E_{p_1} < p_{1a} < 2E_{p_1}$.
5. Calculate $d_{p_1} = d_{p_{1a}} d_{p_{1b}}$ and $p_1 = p_{1a} + 1$. If $p_1$ is not prime, then go to Step 3.
6. Repeat steps 2 to 5 to calculate the primes $p_2, ..., p_x$ and CRT exponents $d_{p_2}, ..., d_{p_x}$.
7. Calculate modulus (N) as $N = p_1 * p_2 * ... * p_x$

Public key is $(e, N)$ and private key $(d_{p_1}, d_{p_2}, ..., d_{p_x}, p_1, p_2, ..., p_x)$

**Correctness of Key Generation Method**

For three prime modulus ($x = 3$), CRT key equations defined for ERRCS are:

$$ed_{p_1} = k_{p_1}(p_1 - 1) + 1 \tag{3.1}$$

$$ed_{p_2} = k_{p_2}(p_2 - 1) + 1 \tag{3.2}$$

$$ed_{p_3} = k_{p_3}(p_3 - 1) + 1 \tag{3.3}$$

Multiplying the equations (3.1), (3.2) and (3.3):

$$(ed_{p_1} - 1)(ed_{p_2} - 1)(ed_{p_3} - 1) = k_{p_1}k_{p_2}k_{p_3}(p_1 - 1)(p_2 - 1)(p_3 - 1) \tag{3.4}$$

or

$$e(e^2 d_{p_1} d_{p_2} d_{p_3} - ed_{p_1} d_{p_2} - ed_{p_1} d_{p_3} - ed_{p_2} d_{p_3} + d_{p_1} + d_{p_2} + d_{p_3}) = k_{p_1} k_{p_2} k_{p_3} \phi(N) + 1 \tag{3.5}$$

It can be written as:

$$ed' = k'\phi(N) + 1 \tag{3.6}$$

Which is the valid RSA key equation (eq. 2.27) with $d' = e^2 d_{p_1} d_{p_2} d_{p_3} - ed_{p_1} d_{p_2} - ed_{p_1} d_{p_3} - ed_{p_2} d_{p_3} + d_{p_1} + d_{p_2} + d_{p_3}$ and $k' = k_{p_1} k_{p_2} k_{p_3}$

Hence the scheme generates a valid pair of public and private exponent.

### 3.1.2 Encryption and Decryption Method

In the proposed method, encryption is done in the same way as in standard RSA; cipher text $C$ is calculated by exponentiation of the message (M) with public exponent ($e$) modulo N (modulus):

$$C = M^e \mod N \tag{3.7}$$

For decryption method due to the use of $x$ primes, the method used in Multi-Prime RSA is used. Message (M) can be calculated from cipher text ($C$ ) computing the intermediate small messages. Intermediate messages ($M_1, M_2, ..., M_x$) are calculated by exponentiation with small decryption exponents (CRT exponents) $d_{p_1}, d_{p_2}, ...d_{p_x}$:

For $1 \leq i \leq x$;

$$C_i = C \mod p_i \tag{3.8}$$

$$M_i = C_i^{d_{p_i}} \mod p_i \tag{3.9}$$

$M$ is computed by combining the intermediate values ($M_i$) using CRT (subsection 2.1.6).

## 3.2 Security Analysis

Several known attacks are studied and the proposed scheme is analyzed against those attacks to prove its security. Small public exponent attacks described for standard RSA are also applicable for ERRCS. Private exponents are taken with the restriction $n_d \geq 2m$ [63], here $m$ is the security parameter and its value is considered as $m = 80$ (for modulus size $n = 1024$ bits); i.e. $n_d \geq 160$ bits. Like RRCS, ERRCS is based on parameter selection and the scheme is designed using multiple primes. Security of ERRCS is analyzed to handle both these aspects; factorization attack due to multiple small primes and parameter selection attack. The proposed scheme is assumed to be free from side channel attack. In this analysis few bits are considered to be known by exhaustive search. The scheme can be further analyzed for partial key exposure attack, by knowing more bits with side channel attack.

### 3.2.1 Factorization Attack

Modulus can be factored probabilistically with known private key $d$ or multiple of $\phi(N)$ [70]. The scheme must be using the modulus large enough to resist general purpose factorization method (NFS) and primes large enough to resist special purpose factorization methods (ECM). Time complexity of NFS increases with modulus size (1024 bits, 2048 bits, 4096 bits and so on). In case of ECM, complexity decreases with increase in number of primes with same size of the modulus. Number of primes can be taken 3,3,4,5 as safe for modulus size 1024, 2048, 4096, 8192 bits respectively [71]. Thus, for 1024 bit and 2048 bit modulus, use of three primes in the proposed scheme is quite secure.

### 3.2.2 Parameter Selection Attack (Lattice method)

Many attacks are proved for two CRT exponents [58],[59]. For the known values of $k_{p_i}$ (in equation (3.1), (3.2), (3.3) ), one must select the public exponent with bound $n_e < [(x-1)/x^2]n - m$ for $x$ primes [59]. For three prime modulus this bound becomes $n_e < 2n/9 - m$ (where $m = 80$ for 1024 bit modulus).

For unknown values of $k_{p_i}$, many factors are considered to prove the security of the proposed schemes. Reconsidering the equations (3.1), (3.2), (3.3) and multiplying:

$$\begin{aligned} &e^3 d_{p_1} d_{p_2} d_{p_3} - e^2[d_{p_1} d_{p_2}(k_{p_3}-1) + d_{p_1} d_{p_3}(k_{p_2}-1) + d_{p_2} d_{p_3}(k_{p_1}-1)] \\ &+e[d_{p_1}(k_{p_2}k_{p_3} - k_{p_3} - k_{p_2} + 1) + d_{p_2}(k_{p_1}k_{p_3} - k_{p_3} - k_{p_1} + 1) + d_{p_3}(k_{p_1}k_{p_2} - k_{p_2} - k_{p_1} + 1)] \\ &- k_{p_1}k_{p_2}k_{p_3}(N-1) - k_{p_1}k_{p_2} - k_{p_2}k_{p_3} - k_{p_1}k_{p_3} + k_{p_1} + k_{p_2} + k_{p_3} - 1 = 0 \end{aligned} \quad (3.10)$$

Rewriting:

$$\begin{aligned} &e^3 d_{p_1} d_{p_2} d_{p_3} - e^2[d_{p_1} d_{p_2}(k_{p_3}-1) + d_{p_1} d_{p_3}(k_{p_2}-1) + d_{p_2} d_{p_3}(k_{p_1}-1)] \\ &+ e[d_{p_1}(k_{p_2}k_{p_3} - k_{p_3} - k_{p_2} + 1) + d_{p_2}(k_{p_1}k_{p_3} - k_{p_3} - k_{p_1} + 1) + d_{p_3}(k_{p_1}k_{p_2} - k_{p_2} \\ &- k_{p_1} + 1)] - k_{p_1}k_{p_2}k_{p_3}(N-1) + k_m + k_0 = 0 \end{aligned} \quad (3.11)$$

In this equation, $d_{p_1}, d_{p_2}, d_{p_3}, k_{p_1}, k_{p_2}, k_{p_3}, k_0$ are unknown variables. Parameter $k_m$ can be found by exhaustive search. Lattice attack can be mounted on this equation by several ways. This equation can be viewed as a linear equation in four variable with ( mod $e^3$), three variables with ( mod $e^2$), two variables with ( mod $e$) and four variables with ( mod $N$). The linear modular polynomial can be solved for the roots by Coppersmith method (sub section 2.1.8).

**Lattice method by modulo $e^3$**

Considering the equation (3.11) modulo $e^3$, linear modular equation in four variables is obtained. The polynomial equation is:

$$f(x,y,z,w) \mod e^3 = e^2x + ey - (N-1)z + w + k_m \quad (3.12)$$

Roots of this polynomial can be defined as $x_0, y_0, z_0, w_0$; where

$$x = d_{p_1} d_{p_2}(1 - k_{p_3}) + d_{p_1} d_{p_3}(1 - k_{p_2}) + d_{p_2} d_{p_3}(1 - k_{p_1})$$

$$y = d_{p_1}(k_{p_2}k_{p_3} - k_{p_3} - k_{p_2} + 1) + d_{p_2}(k_{p_1}k_{p_3} - k_{p_3} - k_{p_1} + 1) + d_{p_3}(k_{p_1}k_{p_2} - k_{p_2} - k_{p_1} + 1)$$

$z = k_{p_1} k_{p_2} k_{p_3}$

$w = k_0$

Polynomial can be soved for the upper bound $X \geqslant x_0, Y \geqslant y_0, Z \geqslant z_0, W \geqslant w_0$, provided that $XYZW < e^3$. Upper bounds are defined as :

$X = 2^{n_e + 3n_d - n/3}$

$Y = 2^{3n_d + 2n_e - 2n/3}$

$Z = 2^{3n_e + 3n_d - n}$

$W = 2^{2n_e + 2n_d - 2n/3 - m}$

Solving for $XYZW < e^3$ with above bounds:

$$11n_d + 5n_e < 8/3n + m \tag{3.13}$$

Substituting the value of $n_d$ from the CRT key equation (2.46); $n_e + n_d = n_k + n/3$; the above inequality (3.13) can be rewritten as:

$$n_k < (6n_e - n + m)/11 \tag{3.14}$$

Using this inequality, only $k_{p_1} k_{p_2} k_{p_3}$ and $k_0$ can be found and substituting these values in equation (3.11), the new equation with variable $d_{p_1} d_{p_2} d_{p_3}$ is obtained, thus we get 5 equations in 6 unknown variables; that cannot be solved. But from above inequality, the resultant value of the term $k_{p_1} k_{p_2} k_{p_3}$ can be found. If one can factor $k_{p_1} k_{p_2} k_{p_3}$, the values of $k_{p_1}, k_{p_2}, k_{p_3}$ can be calculated but with very high complexity of factorization method.

**Lattice method by modulo $e^2$**

Considering the equation (3.11) modulo $e^2$, the equation in three variables is obtained; i.e.

$$f(x, y, z) \mod e^2 = ex + (N-1)y + z + k_m \tag{3.15}$$

Roots of this polynomial can be defined as $x_0, y_0, z_0$; where

$x = d_{p_1}(k_{p_2} k_{p_3} - k_{p_3} - k_{p_2} + 1) + d_{p_2}(k_{p_1} k_{p_3} - k_{p_3} - k_{p_1} + 1) + d_{p_3}(k_{p_1} k_{p_2} - k_{p_2} - k_{p_1} + 1)$

$y = k_{p_1} k_{p_2} k_{p_3}$

$z = k_0$

Linear modular polynomial can be solved for the upper bounds $X, Y, Z$; $X \geqslant x_0, Y \geqslant y_0, Z \geqslant z_0$, provided that $XYZ < e^2$. Upper bounds are defined as :

$X = 2^{3n_d+2n_e-2n/3}$

$Y = 2^{3n_e+3n_d-n}$

$Z = 2^{2n_e+2n_d-2n/3-m}$

Soving for $XYZ < e^2$ using the upper bounds:

$$8n_d + 5n_e < 7/3n + m \tag{3.16}$$

Substituting the value of $n_d$, the above inequality (3.16) can be rewritten as:

$$n_k < (9n_e - n + 3m)/24 \tag{3.17}$$

With this inequality also, no information is revealed about the unknown variables in equation (3.11) except $k_{p_1}k_{p_2}k_{p_3}$.

**Lattice method by modulo** $e$

By using the equation (3.11) modulo $e$, one can obtain the following polynomial:

$$f(x, y) \mod e = (N-1)x + y + k_m \tag{3.18}$$

Roots of this polynomial can be defined as $x_0, y_0$; where

$x = k_{p_1}k_{p_2}k_{p_3}$

$y = k_0$

Linear modular polynomial can be solved for the upper bounds $X, Y$; $X \geqslant x_0, Y \geqslant y_0$, provided that $XY < e$. Upper bounds are defined as :

$X = 2^{3n_e+3n_d-n}$

$Y = 2^{2n_e+2n_d-2n/3-m}$

Soving for $XY < e$ using the upper bounds, one can find the following condition to solve the modular equation:-

$$5n_d + 4n_e < 5/3n + m \tag{3.19}$$

Substituting the value of $n_d$, the above inequality can be rewritten as:

$$n_k < (n_e + m)/5 \tag{3.20}$$

This inequality does not contribute in finding any of the unknown parameters from equation (3.11) except $k_{p_1}k_{p_2}k_{p_3}$.

**Lattice method by modulo $N$**

Considering the equation (3.11) modulo $N$, the following polynomial is obtained:

$$f(x, y, z, w) \mod N = e^3x + e^2y + ez + w + k_m \tag{3.21}$$

Roots of this polynomial can be defined as $x_0, y_0, z_0, w_0$; where

$x = d_{p_1}d_{p_2}d_{p_3}$

$y = d_{p_1}d_{p_2}(k_{p_3} - 1) + d_{p_1}d_{p_3}(k_{p_2} - 1) + d_{p_2}d_{p_3}(k_{p_1} - 1)$

$z = d_{p_1}(k_{p_2}k_{p_3} - k_{p_3} - k_{p_2} + 1) + d_{p_2}(k_{p_1}k_{p_3} - k_{p_3} - k_{p_1} + 1) + d_{p_3}(k_{p_1}k_{p_2} - k_{p_2} - k_{p_1} + 1)$

$w = k_{p_1}k_{p_2}k_{p_3} + k_0$

Polynomial can be soved for the upper bound $X \geqslant x_0, Y \geqslant y_0, Z \geqslant z_0, W \geqslant w_0$, provided that $XYZW < N$. Upper bounds are defined as :

$X = 2^{3n_d}$

$Y = 2^{n_e+3n_d-n/3}$

$Z = 2^{3n_d+2n_e-2n/3}$

$W = 2^{3n_e+3n_d-n}$

Solving for $XYZW < N$ with above bounds, one can solve for the roots if

$$12n_d + 6n_e < 3n \tag{3.22}$$

or

$$n_k < (6n_e - n)/12 \tag{3.23}$$

The value of $d_{p_1}d_{p_2}d_{p_3}$ can be found with the above mentioned bound. Parameters $d_{p_1}$, $d_{p_2}$ and $d_{p_3}$ can be found if factorization of $d_{p_1}d_{p_2}d_{p_3}$ is possible (high complexity of factorization).

Thus, the below mentioned inequality must be considered for the security of the proposed schemes;

1. $11n_d + 5n_e > 8/3n + m$ or $n_k > (6n_e - n + m)/11$
2. $8n_d + 5n_e > 7/3n + m$ or $n_k > (9n_e - n + 3m)/24$
3. $5n_d + 4n_e > 5/3n + m$ or $n_k > (n_e + m)/5$
4. $12n_d + 6n_e > 3n$ or $n_k > (6n_e - n)/12$
5. $n_e < [(x-1)/x^2]n - m$
6. $n_d > 2m$
7. $n_d + n_e = n_k + n/3$

First four conditions can be considered with high complexity of factorization methods. Fifth inequality is for known values of $k_{p_i}$ and sixth is for small CRT. The last equation is from the CRT key equation used in the proposed method.

### 3.2.3 Security Summary

As discussed above, none of the inequalities from (3.13) to (3.22) reveal any important information when the security parameters $k_{p_i}$ are not known. With these bounds the term $k_{p_1}k_{p_2}k_{p_3}$ can be computed and further the parameters $k_{p_1}, k_{p_2}$ or $k_{p_3}$ can be obtained by using factorization method. But this result is obtained with the complexity of lattice method as well as factorization method. Still to maintain the security of the proposed scheme, parameters' values are considered according to the above mentioned inequalities.

Performance of encryption side can be increased by considering the small size of the public exponent and large size of CRT exponent. If one wants performance

gain at decryption side, the public exponent needs to be of large size. Thus, one can adjust the parameter values accordingly. Both the sides can be balanced by selecting the appropriate size of the exponents.

For modulus size $N = 1024$ bits and known values of $k_{p_i}$, $n_e$ can be very small (less than 80 bits) and maximum 147 bits. According to these parameters for small value of public exponent ($n_e = 17$bits), the CRT exponents can be as large as 341 bits (like in MultiPrime RSA). For the maximum value of public exponent ($n_e = 147$bits) the CRT exponent can be 211 bits.

If the CRT exponent is required to be small ($n_d = 160$bits), public exponent ($n_e$) must be greater than 147 bits. For this value, $k_{p_i}$ must be secure (unknown). For unknown values of $k_{p_i}$, the CRT exponent can be 160 bits for $n_e = 280$ bits.

For the proposed scheme, the parameters can be considered as:

In RRCS($n_e < n/2$)[58] results are shown for $n_e = 170$ bits and $n_d = 358$ bits. For maintaining the same encryption speed, the proposed scheme can be compared with RRCS($n_e < n/2$) using parameters $n_e = 170$ bits, $n_d = 280$ bits.

## 3.3 Complexity Analysis

From the key generation method, except prime number generation, all the steps are containing simple computations of addition and multiplication. Complexity in the algorithm is largely dependent upon the random prime number generation and primality testing. Complexity of generating the random prime number is $O(n_p)$, but for ERRCS this complexity will be more as the strict condition of random number generation is involved in the method. Complexity of primality testing is $O(n_p^3)$. Here, the size of the prime is $n/x$, where $n$ is the size of the modulus and $x$ is the number of primes to be generated.

**Encryption Method**

Complexity of encryption method depends on the exponentiation operation (i.e. $M^e \mod N$) by using square and multiply method. In this method, modular square operation is done for every bit in the exponent and modular multiply operation is done for every 1-bit in the exponent. In the proposed method, number of bits in the public exponent $e$ (for $e = 2^{169} + 1$) are taken as ($n_e =$) 170 . For

square and multiply method, 169 square and one multiply operations are involved in the calculation. Hence, for $n$ bit modulus, the encryption complexity of the proposed scheme is:

$$Enc(C) = 170n^2 \tag{3.24}$$

**Decryption Method**

In the proposed scheme, $x$ number of primes are considered. In the key generation method $x$ ($n_d-$bit) decryption exponents (CRT exponents) are computed and hence $x$ small exponentiation operations are carried out using square and multiply method. Size of the decryption exponent ($n_d$) is taken as 280 bits. Using square and multiply method, 280 ($n_d$) square operations and 140 ($n_d/2$) multiply operations are performed for $x$ times and then all these results are combined using CRT. Thus, for $x$ exponentiation; the decryption complexity is (ignoring the computation for CRT),

$$Dec(C) = x * (n_d + 1/2n_d) * n_p^2 \tag{3.25}$$

For $x = 3$, $n_d = 280$ and $n_p = n/3$;

$$Dec(C) = 140n^2 \tag{3.26}$$

**Complexity in Parallelization**

The encryption process do not get any performance gain when implemented in parallel. But the decryption speed enhances if it is run using parallel processors. Using single processor, $x$ computations are done sequentially without parallelization. But due to parallel processing the complexity of decryption process becomes:

$$Dec(C) = (n_d + 1/2n_d) * n_p^2 \tag{3.27}$$

or

$$Dec(C) = 47n^2 \tag{3.28}$$

In this calculation also, the complexity of combining the intermediate results (CRT) is ignored.

## 3.4 Implementation Details

The proposed scheme is implemented in C++ on windows operating system. In this scheme, the size of the modulus can be 1024 bits, 2048 bits or even more than this. To deal with such a big data, special type of library files are needed. NTL with GMP [101] (Appendix A) is very popular library among RSA researchers. NTL is Number Theory Library and GMP is GNU Multi-Precision library. GMP is used with NTL for extra speedup. For basic operations (e.g. multiplication), GMP increases the speed of NTL by a factor of 2 or more. For the installation of GMP, Cygwin is used on Windows operating system.

For implementing the proposed scheme, NTL with GMP is used on a laptop with 2.3 GHz CPU and 8 GB RAM. Big number is dealt with the header file ZZ.h. For the Extended Euclidean method, XGCD() is used to find the parameters in key generation method and primality of the integer is tested using MillerWitness() function. For the exponentiation operation PowerMod() is used in encryption and decryption method.

Key generation, encryption and decryption methods of the proposed scheme are executed for 100 times and average is computed to store the results. In table 3.1, the resultant values are shown in different combinations of the exponents $(n_e, n_d)$ for modulus size 1024 bits.

Following is the result of the implementation for unknown $k_{p_i}$ (109 bits) with modulus size N=1024 bits:

Size($p_1$)=340bits

Size($p_2$)=340bits

Size($p_3$)=341 bits

Size($e$)=170 bits

Size($d_{p_1}$)=280 bits

Size($d_{p_2}$)=280 bits

Size($d_{p_3}$)=280 bits

Size($k_{p_1}$)=109 bits

Size($k_{p_2}$)=109 bits

Size($k_{p_3}$)=109 bits

For known $k_{p_i}$, values obtained for different parameters are as follows:

Size($N$)=1020 bits

Size($p_1$)=340 bits

Size($p_2$)=340 bits

Size($p_3$)=341 bits

Size($e$)=17 bits

Size($d_{p_1}$)=339 bits

Size($d_{p_2}$)=340 bits

Size($d_{p_3}$)=340 bits

Size($k_{p_1}$)=16 bits

Size($k_{p_2}$)=16 bits

Size($k_{p_3}$)=16 bits

TABLE 3.1: Implementation results of the proposed scheme for N=1024bits

| $n_e$ | $n_d$ | Random Numbers generated | Key-Gen Time (ms) |
|---|---|---|---|
| 280 | 170 | 2541 | 115.95 |
| 200 | 250 | 2722 | 123.103 |
| 170 | 280 | 2485 | 115.26 |
| 140 | 217 | 2773 | 118.99 |
| 100 | 257 | 2900 | 117.401 |
| 17 | 340 | 2495 | 104.72 |

In table 3.1, implementation results are shown for unknown $k_{p_i}$ (first three rows) and for known $k_{p_i}$ (last three rows).

Proposed scheme is also implemented with 2048 bits modulus. For known $k_{p_i}$, public exponent is taken to be small, i.e. $n_e < 327$. For unknown $k_{p_i}$ (130 bits), CRT exponents are taken to be small. The parameter $m$ is considered to be of 128 bits in this case. The implementation results for different combinations of $n_e$ and

$n_d$ with modulus size 2048 bits are shown in table 3.2. Details of the examples are given in Appendix B.1.

For unknown $k_{p_i}$, following values are obtained:

Size($N$)=2048 bits

Size($p_1$)=682 bits

Size($p_2$)=683 bits

Size($p_3$)=683 bits

Size($e$)=551 bits

Size($d_{p_1}$)=262 bits

Size($d_{p_2}$)=263 bits

Size($d_{p_3}$)=263 bits

Size($k_{p_1}$)=130 bits

Size($k_{p_2}$)=130 bits

Size($k_{p_3}$)=130 bits

For known $k_{p_i}$, the following values are obtained:

Size($N$)=2045 bits

Size($p_1$)=681 bits

Size($p_2$)=682 bits

Size($p_3$)=683 bits

Size($e$)=17 bits

Size($d_{p_1}$)=681 bits

Size($d_{p_2}$)=682 bits

Size($d_{p_3}$)=683 bits

Size($k_{p_1}$)=16 bits

Size($k_{p_2}$)=16 bits

Size($k_{p_3}$)=16 bits

TABLE 3.2: Implementation results of the proposed scheme for N=2048bits

| $n_e$ | $n_d$ | Random Number Generated | Key Gen Time (ms) |
|---|---|---|---|
| 551 | 262 | 4484 | 1074.19 |
| 400 | 413 | 5350 | 1256.67 |
| 300 | 513 | 5444 | 1312.34 |
| 200 | 499 | 5265 | 1253.70 |
| 100 | 599 | 5996 | 1353.09 |
| 17 | 682 | 4174 | 1115.75 |

In table 3.2, first three rows show the implementation results for unkown $k_{p_i}$ and last three rows for known $k_{p_i}$.

Efficiency of the key generation method is directly dependent upon the size of the modulus. Prime random numbers are generated with complexity $O(n)$, where $n$ is the size of the modulus. Efficiency of the key generation method can be analysed by the number of primes generated to get the key pair of the proposed scheme. Due to the strict conditions, random numbers generated are almost 2 to 3 times the modulus size (table 3.1 and 3.2).

## 3.5 Comparative Analysis

Table 3.3 (figure 3.1) gives the complexity comparison of the proposed scheme with other variants of RSA cryptosystem. The proposed scheme (ERRCS) is directly influenced by only a few RSA variants like RSA CRT, MultiPrime RSA, Rebalanced RSA and RRCS. Thus only these variants are considered for the efficiency demonstration of ERRCS. For simplicity, the bit-size of the modulus ($n$) is considered to be 1024 bits.

Exponentiation operations in encryption and decryption methods are calculated by the square and multiply method; chinese remainder theorem is used to calculate the complexity of decryption method.

ERRCS is more efficient than RRCS as the scheme is using less number of bits for the decryption exponent (table 3.3). In ERRCS, for three prime modulus,

TABLE 3.3: Complexity comparison of the Proposed Scheme with existing RSA Variants

| Sr. No. | RSA Variants | Public exponent (bits) $n_e$ | Encryption Complexity | Decryption Exponent (bits) $n_d$ | Decryption Complexity |
|---|---|---|---|---|---|
| 1. | RSA[4] | 17 | $17n^2$ | 1024 | $1536n^2$ |
| 2. | RSA CRT[54] | 17 | $17n^2$ | 512 | $384n^2$ |
| 3. | MultiPrime RSA[56] | 17 | $17n^2$ | 342 | $170n^2$ |
| 4. | Rebalanced RSA[31] | 1024 | $1536n^2$ | 160 | $120n^2$ |
| 5. | RRCS ($n_e < n/2$) [58] | 170 | $170n^2$ | 358 | $268n^2$ |
| 6. | ERRCS | 170 | $170n^2$ | 280 | $140n^2$ |

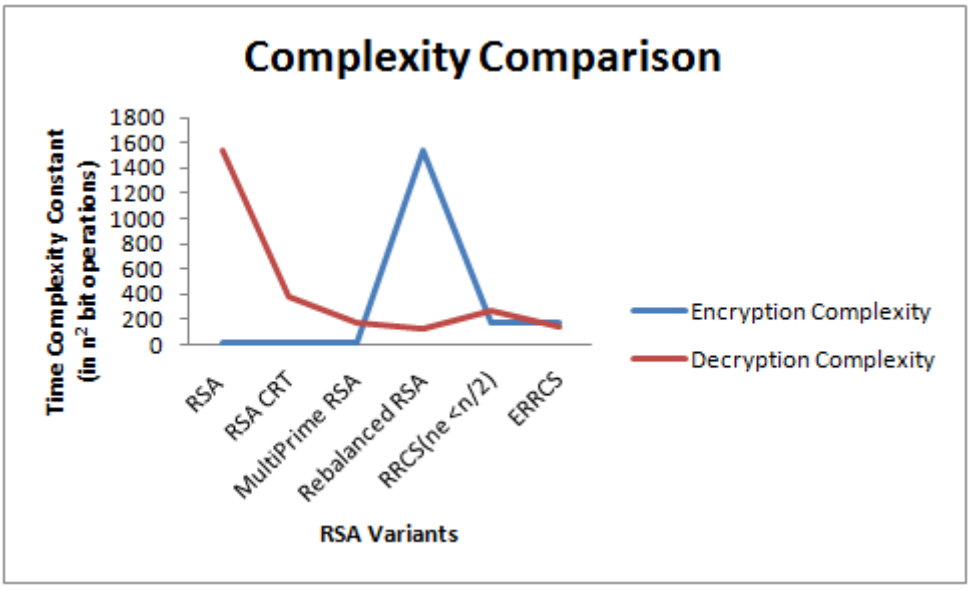


FIGURE 3.1: Complexity comparison of the Proposed Scheme with existing RSA Variants

$n_d = 250$ and in RRCS for two prime modulus $n_d = 358$ bits. Performance of ERRCS (in terms of decryption speed) increases with the use of small bit size of CRT exponents.

Complexity of all the variants of RSA mentioned in table (3.3) is calculated using single processing environment. If the variants are implemented using parallel processing environment, better performance (in terms of decryption complexity) will be exhibited.

Performance of ERRCS is better as compared to other RSA variants as shown

in table (3.3). ERRCS is better than basic RSA, RSA CRT, MultiPrime RSA in terms of decryption speed. Decryption speed of ERRCS is 10.9 times, 2.7 times and 1.2 times better than basic RSA, RSA CRT and MultiPrime RSA respectively. Encryption speed of ERRCS is 9.1 times better as compared to Rebalanced RSA, but the decryption speed is 1.2 times slower. In this work RRCS is enhanced to get the better decryption speed. For the same size of public exponent, decryption speed of ERRCS is improved by a factor of 1.9 as compared to RRCS ($n_e < n/2$).

ERRCS and other variants are implemented using NTL with GMP with 2.3 GHz CPU and 8 GB RAM. Table 3.4 shows the comparison of the implementation results of ERRCS with the existing RSA variants.

TABLE 3.4: Comparison of the proposed scheme with existing RSA variants: implementation

| Sr. No. | RSA Variant | Average Enc Time (ms) | Std Deviation (Enc Time ms) | Average Dec Time (ms) | Std Deviation (Dec Time ms) |
|---|---|---|---|---|---|
| 1. | RSA | 0.0906 | 0.0115 | 3.841 | 0.208 |
| 2. | RSA CRT | 0.094 | 0.004 | 1.25 | 0.194 |
| 3. | MultiPrime RSA | 0.098 | 0.00904 | 0.62 | 0.035 |
| 4. | Rebalanced RSA | 3.827 | 0.304 | 0.46 | 0.02254 |
| 5. | RRCS($n_e < n/2$) | 0.60 | 0.093 | 0.88 | 0.141 |
| 6. | ERRCS | 0.62 | 0.112 | 0.47 | 0.198 |

It is clear from table 3.4 (or figure 3.2), that ERRCS is more efficient than RRCS ($n_e < n/2$) scheme. For same encryption speed, decryption speed of the proposed scheme is approximately 1.8 times faster as compared to RRCS scheme.

In implementation, gain in decryption speed is less than the gain in theoretical calculation. This is because of the added complexity of simple computations; like CRT (used to combine the intermediate results).

As the results shown in Table 3.4 are implemented outcome of the variants on a system with 2.3 GHz CPU and 8 GB RAM, the variants are also implemented using other computational environment so as to exhibit the performance in a better way. The results in Table 3.5 are the implementation outcome of the variants on cPCI(compact Peripheral Component Interconnect) architecture with 1.6 GHz CPU and 4 GB RAM. Time taken by encryption and decryption process on cPCI architecture is more than the results shown in table 3.4 due to the constrained

TABLE 3.5: Comparison of proposed scheme with existing RSA variants implemented on cPCI Architecture

| Sr. No. | RSA Variant | Average Enc Time (ms) | Std Deviation (Enc Time ms) | Average Dec Time (ms) | Std Deviation (Dec Time ms) |
|---|---|---|---|---|---|
| 1. | RSA | 0.199219 | 0.02435 | 5.082333 | 1.23067 |
| 2. | RSA CRT | 0.194603 | 0.02314 | 1.724625 | 0.340456 |
| 3. | MultiPrime RSA | 0.217 | 0.03461 | 0.97861 | 0.092341 |
| 4. | Rebalanced RSA | 5.204617 | 1.67483 | 0.661424 | 0.053610 |
| 5. | RRCS($n_e < n/2$) | 0.708578 | 0.0290378 | 1.210942 | 0.834011 |
| 6. | ERRCS | 0.722391 | 0.053902 | 0.867 | 0.0431009 |

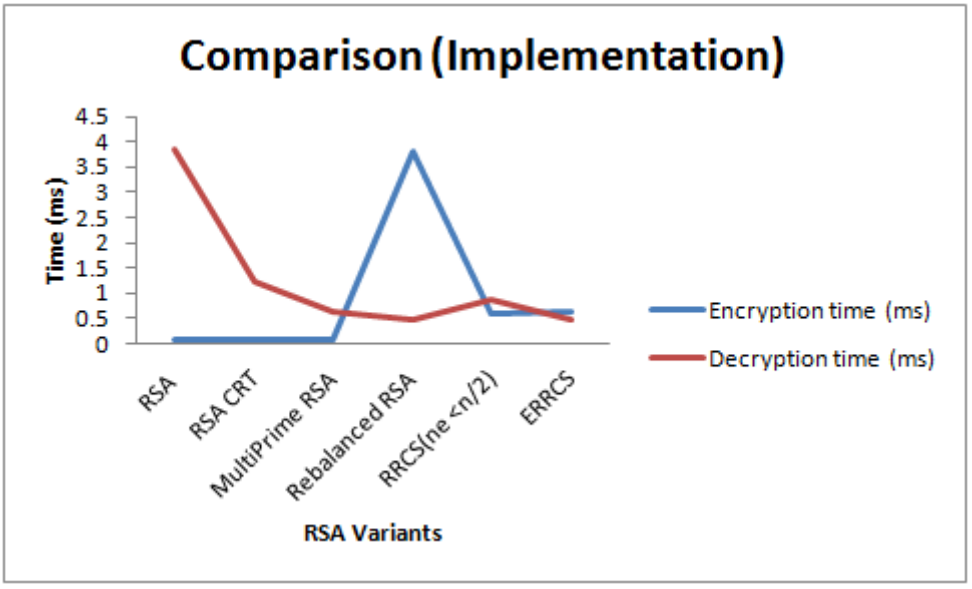


FIGURE 3.2: Comparison of the Proposed Scheme with existing RSA Variants: Implementation

environment (less computational speed) used in the architecture. The performance gain in table 3.5 is approximately same as in table 3.4.

## 3.6 Conclusion

As many applications have the requirement to balance both the encryption and decryption sides (communication between small devices, like smart phones), RRCS balances both the encryption and decryption sides. Proposed scheme (ERRCS) further improves the decryption side while maintaining the same encryption speed

as in RRCS. Theoretically, the decryption speed of ERRCS increases by a factor of 1.9 and in implementation, speed increases by a factor of 1.8.

ERRCS scheme offers advantage in both encryption as well as decryption speed in comparison with RRCS. One can get performance gain in encryption or decryption sides by adjusting the security parameters $(n_e, n_d)$. Security of the proposed scheme is proved to be more as compared to RRCS. As MultiPrime RSA variant has already been an active variant of RSA, this scheme is feasible in any RSA embedded protocol.

# Chapter 4

# Improvement in Dual RSA

In previous chapter, improvement is done in computational performance of RSA variant (RRCS). In this chapter, another RSA variant (Dual RSA) is considered for the improvement which is based on the efficient use of memory consumption. Dual RSA is based on the reduction in memory consumption for two RSA instances. In this chapter, three schemes are proposed which offer improvement over three existing schemes of Dual RSA (Small-e, Small-d and Dual Generalized Rebalanced RSA) in a sense that the proposed schemes result in fast computations as well as less memory consumption. Proposed schemes for resource constrained (RC) environment are RC RSA-I, RC RSA-II and RC RSA-III. In RC RSA-I, multiple prime modulus is used instead of two prime modulus(as in Dual RSA Small-e). As compared to Dual RSA Small-e, the decryption speed of RC RSA-I becomes 9 times faster. In Dual RSA Small-d, the encryption exponent is very large due to which the encryption method is very time consuming. In RC RSA-II, Dual RSA Small-d is improved by performing the time consuming encryption computation offline when the computing device is relatively free. Due to this, the encryption time becomes almost negligible. Dual Generalized Rebalanced RSA (DGRR) is based on the selection of encryption and decryption exponents. In RC RSA-III, DGRR is extended to improve the decryption speed by using multiple primes. In RC RSA-III, the decryption speed becomes 2.7 times faster as compared to DGRR. RC RSA-I, RC RSA-II and RC RSA-III are elaborated with security analysis, complexity analysis and implementation details in section 4.1, 4.2 and 4.3 respectively. Proposed schemes are compared with other RSA variants in section 4.4.

## 4.1 RC RSA-I

In Dual RSA Small-e [61], keys are calculated such that two instances (two different moduli) can share the same public and private exponents. Use of common exponents for two different moduli makes the memory consumption by two RSA instances less. Dual RSA Small-e is further improved here to increase the decryption speed.

Unlike Dual RSA Small-e, where two primes are used, in RC RSA-I multiple primes are used. As in Dual RSA Small-e, two instances in RC RSA-I with different multiple prime modulus share common public and private exponents.

Proposed method is based on the following two RSA key equations:

$$ed = 1 + k_1 * (p_1 - 1)(p_2 - 1)...(p_x - 1) \tag{4.1}$$

$$ed = 1 + k_2 * (q_1 - 1)(q_2 - 1)...(q_x - 1) \tag{4.2}$$

($x$ is the number of primes considered for the modulus $N$.)

Proposed method generates two RSA instances such that both share common public and private exponents. The parameters $k_1, k_2, p_1, p_2, ..., p_x$ and $q_1, q_2, ..., q_x$ are calculated so as to satisfy these RSA key equations. Following are the details of key generation method:

### 4.1.1 Key Generation Method

In the algorithm, $n_e$=number of bits in $e$, $n_d$=number of bits in $d$, $n$ =number of bits in moduli ($N_1$ and $N_2$), $n_k$=number of bits in security parameters ($k_1$ and $k_2$), $n/x$=number of bits in primes $p_{1_1}, p_{1_2}, ...p_{1_x}, p_{2_1}, p_{2_2}, ...p_{2_x}$.

1. Public exponent $e$ is selected as small random integer, say $n_e < n/x$. It might be sparse integer to have a less computational cost.

2. Prime factors $p_{1_1}, p_{1_2}, ...p_{1_x}, p_{2_1}, p_{2_2}, ...p_{2_x}$ are computed as $p_{1_1} = u_{1_1}u_{2_1} + 1, p_{2_1} = u_{1_1}u_{2_2} + 1, p_{1_2} = u_{1_2}u_{2_2} + 1, p_{2_2} = u_{1_2}u_{2_3} + 1, ..., p_{1_{x-1}} = u_{1_{x-1}}u_{2_{x-1}} + 1, p_{2_{x-1}} = u_{1_{x-1}}u_{2_x}$ such that $u_{1_1}, u_{1_2}, ..., u_{1_{x-1}}$ with $n_e$ bit and $u_{2_1}, u_{2_2}, ..., u_{2_x}$ with $(n/x - n_e)$ bit are random numbers satisfying

$GCD(e, u_{1_1}) = GCD(e, u_{1_2}) = ... = GCD(e, u_{1_{x-1}}) = GCD(e, u_{2_1}) = GCD(e, u_{2_2}) = ... = GCD(e, u_{2_{x-1}}) = GCD(e, u_{2_x}) = 1$.

3. Select $n_e$-bit random integer $u_{1_x}$ such that $GCD(e, u_{1_x}) = 1$ and compute $p_{1_x} = u_{1_x} u_{2_x} + 1$.

4. Parameters $d$ and $k_1$ are computed such that $ed = 1 + k_1(p_{1_1} - 1)(p_{1_2} - 1)...(p_{1_x} - 1)$

5. Compute $p_{2_x} = k_1 u_{2_1} + 1$, if $p_{2_x}$ is not prime, then repeat the process by taking another random choice for the calculation of the prime factor $p_{1_x}$ in step 3.

$N_1$ is calculated as $N_1 = p_{1_1} p_{1_2} ... p_{1_x}$ and $N_2$ is calculated as $N_2 = p_{2_1} p_{2_2} ... p_{2_x}$ and $k_1$, $k_2$ are security parameters such that $k_2 = u_{1_x}$.

For modulus size 1024 or 2048 bits, number of primes must be at the most three [71], the algorithm is described below for modulus with 3 prime factors; $N_1 = p_1 q_1 r_1$ and $N_2 = p_2 q_2 r_2$.

**Key Generation Method (with $x = 3$)**

Here, $n_e$=number of bits in $e$ ($n_e < n/3$), $n_d$=number of bits in $d$, $n$ =number of bits in moduli ($N_1$ and $N_2$), $n_k$=number of bits in security parameters ($k_1$ and $k_2$), $n/3$=number of bits in primes ($p_1, q_1, r_1$ and $p_2, q_2, r_2$).

1. Public exponent $e$ is selected as small random integer, say $n_e < n/3$. It might be sparse integer to have a less computational cost.

2. Prime factors $p_1, q_1, p_2, q_2$ are computed as $p_1 = u_1 u_2 + 1, p_2 = u_1 v_2 + 1, q_1 = v_1 v_2 + 1$ and $q_2 = v_1 w_2 + 1$ such that $u_1, v_1$ with $n_e$ bits and $u_2, v_2, w_2$ with $(n/3 - n_e)$ bits are random numbers satisfying $GCD(e, u_1) = GCD(e, v_1) = GCD(e, u_2) = GCD(e, v_2) = GCD(e, w_2) = 1$.

3. Select $n_e$-bit random integer $w_1$ such that $GCD(e, w_1) = 1$ and compute $r_1 = w_1 w_2 + 1$.

4. Parameters $d$ and $k_1$ are computed such that $ed = 1 + k_1(p_1 - 1)(q_1 - 1)(r_1 - 1)$.

5. Compute $r_2 = k_1 u_2 + 1$, if $r_2$ is not prime, then repeat the process by taking another random choice for the calculation of the prime factor $r_1$ in step 3.

Output is: $N_1 = p_1 q_1 r_1$ and $N_2 = p_2 q_2 r_2$, $k_2 = w_1$.

**Correctness of Key Generation Method**

As the RSA key equation is $ed = 1 + k\phi(N)$. The following way will prove the basic RSA key equation for RC RSA-I for three primes:

$$\begin{aligned} ed &= 1 + k_1\phi(N_1) && (4.3)\\ &= 1 + k_1(p_1 - 1)(q_1 - 1)(r_1 - 1) && (4.4)\\ &= 1 + k_1(u_1 u_2)(v_1 v_2)(w_1 w_2) && (4.5)\\ &= 1 + w_1(u_1 v_2)(v_1 w_2)(k_1 u_2) && (4.6)\\ &= 1 + k_2(p_2 - 1)(q_2 - 1)(r_2 - 1) && (4.7)\\ &= 1 + k_2\phi(N_2) && (4.8) \end{aligned}$$

Hence the keys generate valid RSA key equation (eq. 2.27). Similarly, the key generation method for $x$ primes can be proved.

### 4.1.2 Encryption and Decryption Method

Encryption method is implemented by the same method as in standard RSA; cipher texts $C_1$ and $C_2$ are calculated by exponentiation of the messages ($M_1$ and $M_2$) using moduli $N_1$ and $N_2$:

$$C_1 = M_1^e \mod N_1 \tag{4.9}$$

and

$$C_2 = M_2^e \mod N_2 \tag{4.10}$$

(where $e$ is the common public exponent for two RSA exponents.)

As $x$ prime pairs $(p_{1_i}, p_{2_i})$ for $1 \leq i \leq x$ (here $x = 3$) are generated in the proposed scheme, message ($M_1$ and $M_2$) can be derived by the computation of the exponentiation using small decryption exponents (called CRT exponents) $d_1, d_2, ..., d_x$;

$$M_{1_i} = C_1^{d_i} \mod p_{1_i} \tag{4.11}$$

where $d_i = d \mod p_{1_i}$ and

$$M_{2_i} = C_2^{d_i} \mod p_{2_i} \tag{4.12}$$

where $d_i = d \mod p_{2_i}$

$M_1$ and $M_2$ can be computed by combining these intermediate values using CRT (subsection 2.1.6).

### 4.1.3 Security Analysis

As RC RSA-I is based on Dual RSA Small-e and MultiPrime RSA, impact of both are considered here. Existing known security attacks are studied and analyzed. Due to the use of multiple prime modulus, factorization attack becomes very important for RC RSA-I. Besides this, from equations (4.1) and (4.2), security parameters $k_1$ and $k_2$ become critical. Security of RC RSA-I is analyzed for both known and unknown values of the parameters $k_1$ and $k_2$. Due to these security parameters, small encryption exponent ($e$) used in the proposed scheme is discussed. In this analysis, the proposed scheme is considered to be free from side channel attack. Few bits are considered to be known in the scheme by exhaustive search. Considering side channel attack more bits can be known and the scheme can further be analyzed for partial key exposure attack.

#### Factorization Attack

Security of RC RSA-I can be broken by factoring the modulus $N$. Once the private key $d$ or multiple of $\phi(N)$ is known, one can factor the modulus probabilistically [70].

In RC RSA-I, modulus $N$ is considered to be the product of more than two primes. There are many methods by which the composite modulus can be factored; like NFS (Number Field Sieve) and ECM (Elliptic Curve Method). RSA modulus with size 1024 bits, 2048 bits, 4096 bits is considered to be secure with 3, 3, 4 prime factors respectively[71]. Hence in RC RSA-I, for modulus size $N = 1024$ bits and $N = 2048$ bits, three balanced primes can be taken as secure. Security will be decreased if more number of primes are taken under these moduli sizes.

**Unknown $k_1$ and $k_2$ (lattice method)**

By using lattice method, two security parameters $k_1$ and $k_2$ can be revealed if $n_e + n_d < 7n/6 + m/2$ or $n_e < n/6 + m/2$, for $n_d \approx n$. Following is the detail:

RSA key equation $ed = 1 + k\phi(N)$ can be written as

$$\begin{aligned} ed &= 1 + k(p-1)(q-1)(r-1) && (4.13)\\ &= 1 + k(pqr - pq - pr - qr + p + q + r - 1) && (4.14)\\ &= 1 + k(N - (pq + pr + qr - p - q - r + 1)) && (4.15)\\ &= 1 + k(N - t) && (4.16) \end{aligned}$$

Where $t = (pq + pr + qr - p - q - r + 1)$ with size $(n_t =)2n/3$ bits.

Two key equations in RC RSA, $ed = 1 + k_1\phi(N_1)$ and $ed = 1 + k_2\phi(N_2)$, can be written as

$$ed = 1 + k_1(N_1 - t_1) \tag{4.17}$$

and

$$ed = 1 + k_2(N_2 - t_2) \tag{4.18}$$

Taking the difference of these two equations (4.17) and (4.18):

$$k_1(N_1 - t_1) = k_2(N_2 - t_2) \tag{4.19}$$

Few bits (say $m$ bits) of $k_2$ can be known by exhaustive search such that $k_2 = k_l + k_m$, where $k_l$ part is unknown and $k_m$ is known. Equation (4.19) can be rewritten as:

$$k_1N_1 - k_lN_2 + k_lt_2 + k_mt_2 - k_1t_1 - k_mN_2 = 0 \tag{4.20}$$

$$f(x, y, z) = N_1x - N_2y + z - C \tag{4.21}$$

Roots of the polynomial are $x_0, y_0, z_0$; where $x = k_1$, $y = k_l$, $z = k_lt_2 + k_mt_2 - k_1t_1$ and $C = k_mN_2$ (C is constant). The polynomial can be solved for the upper bounds $X \geq x_0$, $Y \geq y_0$, $Z \geq z_0$, provided that N is very large and $XYZ < M$, where $M = ||f(xX, yY, zZ)||_\infty = 2^{n_e+n_d}$. Bounds X, Y, Z are as follows:

$X = 2^{n_e+n_d-n}$, $Y = 2^{n_e+n_d-n-m}$ and $Z = 2^{n_e+n_d-n/3}$

Solving for $XYZ < M$

$$n_e + n_d < 7n/6 + m/2 \tag{4.22}$$

for $n_d \approx n$:

$$n_e < n/6 + m/2 \tag{4.23}$$

Cryptanalysis using continued fractions also give approximately same bound. Thus the two security parameters ($k_1$ and $k_2$) can be revealed if $n_e < 211$ for $n = 1024$, $m = 80$ and $n_d = n$.

**Known $k_1$, $k_2$ and small public exponent**

The effect of small public exponent with known $k_1$ and $k_2$ is shown below: For $N_1 = p_1q_1r_1$ and $N_2 = p_2q_2r_2$, $p_1 = u_1u_2 + 1, q_1 = v_1v_2 + 1, r_1 = w_1w_2 + 1, p_2 = u_1v_2 + 1, q_2 = v_1w_2 + 1, r_2 = k_1u_2 + 1$

In RC RSA-I, $N_1 = p_1q_1r_1$ and $N_2 = p_2q_2r_2$ can be rewritten as:

$$N_1-1 = u_1u_2v_1v_2k_2w_2+u_1u_2k_2w_2+v_1v_2k_2w_2+u_1u_2v_1v_2+k_2w_2+u_1u_2+v_1v_2 \tag{4.24}$$

$$N_2-1 = u_1u_2v_1v_2k_1w_2+u_1u_2k_1v_2+v_1u_2k_1w_2+u_1v_1v_2w_2+k_1u_2+u_1v_2+v_1w_2 \tag{4.25}$$

In this case $k_1$ and $k_2$ are known and $u_1, v_1, w_1$ can be found by exhaustive search. The unknown parameters are $u_2, v_2$ and $w_2$; these three unknown parameters cannot be calculated using two equations only. Thus exhaustive search attack doesn't work for RC RSA-I.

**Known $k_1$ and $k_2$ (Lattice based method)**

In this method, by using Lattice theory, it is proved that known $k_1$ and $k_2$ do not reveal any information about the factorization of RC RSA-I modulus.

Considering again the equation (4.19);

$k_1(N_1 - t_1) = k_2(N_2 - t_2)$

Computing $k_1' = \frac{k_1}{GCD(k_1,k_2)}$ and $k_2' = \frac{k_2}{GCD(k_1,k_2)}$ such that $GCD(k_1', k_2') = 1$

$$k_1'(N_1 - t_1) = k_2'(N_2 - t_2) \tag{4.26}$$

In the above equation, $k_1'$ and $k_2'$ are known; only $t_1$ and $t_2$ are unknown. Taking mod $k_2'$:

$$t_1 \equiv N_1 \mod k_2' \tag{4.27}$$

Assuming $C_1 = N_1 \mod k_2'$; $t_1 = C_1 + \psi k_2'$, here $\psi$ is unknown with size

$n_\psi = 2n/3 - (n_e + n_d - n - \gamma)$ (where $\gamma$ is the bit length of $GCD(k_1, k_2)$)

Substituting the value of $t_1$ in equation (4.26) :

$$k_1'(N_1 - C_1 - \psi k_2') = k_2'(N_2 - t_2) \tag{4.28}$$

Taking this equation Modulo $N_1$:

$$f_{N1}(x, y) = k_1'k_2'x - k_2'y + k_2'N_2 + k_1'C_1 \tag{4.29}$$

For the above equation, $x = \psi$, $y = t_2$, the polynomial can be solved with lattice method for upper bounds $X$, $Y$ and the roots can be found if $XY < N_1$;

$$2n/3 - (n_e + n_d - n - \gamma) + 2n/3 < n \tag{4.30}$$

Simplifying it:

$$n_e > n/3 + \gamma \tag{4.31}$$

It means that all the parameters can be known if the above inequality holds, for $n_e > n/3$. In RC RSA-I, public exponent is always taken as less than $n/3$. Thus, RC RSA-I is secure for low values of public exponent.

**Security Summary of RC RSA-I**

It is proved above that for $n_e < n/6 + m/2$ ($n_e < 211$), $k_1$ and $k_2$ can be computed by lattice method. But the computation of these parameters do not reveal any critical information (about unknown parameters); i.e. RC RSA-I cannot be broken by exhaustive search or factorization of the modulus.

### 4.1.4 Complexity Analysis

In the key generation method, primality testing is done to check for every pair of prime factors $(p_i, q_i)$; for $1 \leqslant i \leqslant x$ instances. Except for primality testing, all the

steps are implemented using small computations of addition and multiplication. For the computation of complexity of key generation method, simple computations are ignored; only the complexity of random prime number and the primality testing is considered. Complexity of generating the random prime number is $O(n_p)$ and the complexity for primality testing is $O(n_p^3)$. In the key generation method of RC RSA-I, small size of the prime factors are considered, thus the complexity to find $x$ prime pairs is less as compared to random prime generation in stanadard RSA.

**Encryption Method**

Proposed scheme, RC RSA-I is designed for two RSA instances ($N_1$ and $N_2$), encryption method for any of the instances is done with exponentiation operation as in standard RSA (using square and multiply method). In RC RSA-I, size of the public exponent $e$ can be taken as very small (say $e = 2^{16} + 1$), thus for square and multiply method, 16 square operations and only one multiply operation is involved in the calculation of encryption method. Hence, for $n$ bit modulus, the encryption complexity of RC RSA-I is:

$$Enc(C) = 17n^2 \tag{4.32}$$

**Decryption Method**

In the proposed scheme, $x$ number of primes are considered. In the key generation method $x$ exponentiation are performed to get $x$ intermediate messages. Size of the decryption exponents (CRT exponents) is computed as $n_d = n/3$ ($n$ is the number of bits in the modulus). Performing $x$ computations of square and multiply method and ignoring the computations involved in CRT, the decryption complexity,

$$Dec(C) = x * (n_d + 1/2n_d)n_p^2 \tag{4.33}$$

For $x = 3$, $n_d = 341$ bits and $n_p = n/3$;

$$Dec(C) = 170n^2 \tag{4.34}$$

**Complexity in Parallelization**

Using parallelization the computations involved in the encryption method will not be affected. When the parallelization of the computations is considered, the decryption process becomes efficient. Computations involved in small exponentiation

operations can be done in parallel, then the decryption complexity results in:

$$Dec(C_p) = (n_d + n_d/2)n_p^2 \tag{4.35}$$

or

$$Dec(C_p) = 57n^2 \tag{4.36}$$

**Memory Usage**

Because of the generation of common public and private exponents for two different instances, memory consumption of RC RSA-I is low. For the computation of memory consumption, three factors are involved: modulus ($N$), public exponent ($e$) and decryption exponent ($d$). Memory consumption for two RSA instances is; $2n$ bits for two modulus ($N_1$ and $N_2$), 17 bits for encryption exponent ($e$) and $n$ bits for the decryption exponent ($d$). Memory usage by RC RSA-I is;

$$Mem(C) = (3n + 17)bits \tag{4.37}$$

Hence, the complexity is:

$$\begin{aligned} Enc(C) &= 17n^2 \\ Dec(C) &= 170n^2 \\ Mem(C) &= (3n + 17)bits \end{aligned}$$

### 4.1.5 Implementation Details

RC RSA-I is implemented in C++ on windows operating system. The scheme is implemented with modulus size 1024 bits and 2048 bits. To handle the computations involved with these big data, special type of library files, NTL with GMP [101] (Appendix A) is used, which is generally used by RSA researchers. NTL is the Number Theory Library and GMP is GNU Multi-Precision library. NTL is containing library functions which efficiently handle the big computations. The code is written using simple commands of C++, which is easy to understand. In this work the focus is on the performance of RSA computations, so GMP is used with NTL for extra speedup. For basic operations (e.g. multiplication) GMP increases the speed of NTL by a factor of 2 or more. GMP runs with NTL using

Unix based system. Thus to use GMP, Cygwin is used on Windows operating system. Here, the proposed schemes are implemented on laptop with 2.3 GHz CPU and 8 GB RAM, using NTL with GMP.

For the implementation of the proposed scheme the basic header file ZZ.h is used. For the Extended Euclidean method, XGCD() is used to find the parameters in the key generation method and primality of the integer is tested using MillerWitness() function. For the exponentiation operation PowerMod() is used in encryption and decryption method.

Proposed scheme is implemented with key generation, encryption and decryption methods and executed for 100 times to find the average values. Details of the examples are given in appendix B.2.

With modulus size 1024 bits, the following values are recorded:

Size($N_1$)=1024 bits

Size($N_2$)=1024 bits

Size($d$)=1024 bits

Size($e$)=17 bits

Time consumed by key generation algorithm=123.812 ms

Time consumed by encryption=0.0925 ms

Time consumed by decryption=0.627 ms

TABLE 4.1: Implementation Results of RC RSA-I for N=1024 bits

| $n_e$ | Random Number Generated | Key Generation Time (ms) |
|---|---|---|
| 200 | 815 | 129.97 |
| 150 | 731 | 136.242 |
| 100 | 796 | 136.573 |
| 50 | 835 | 127 |
| 17 | 762 | 123.812 |

The algorithm is also implemented with modulus size 2048 bits, and the following values are obtained:

bits in $N_1$=2047

bits in $N_2$=2047

bits in $d$=2048

bits in $e$=17

Time consumed by key generation algorithm=1433.01 ms

Time consumed by encryption=0.25 ms

Time consumed by decryption=3.74 ms

TABLE 4.2: Implementation Results of RC RSA-I for N=2048 bits

| $n_e$ | Random Number Generated | Key Generation Time (ms) |
|---|---|---|
| 400 | 1518 | 1346.1 |
| 300 | 1591 | 1499.27 |
| 200 | 1543 | 1418.26 |
| 100 | 1549 | 1415.6 |
| 17 | 1597 | 1433.01 |

Efficiency of the algorithm depends on the size of the modulus. From table (4.1 and 4.2), random number generated are of the order of the size of the moduli (1024 bits in table (4.1) and 2048 bits in table (4.2)). The values are rather less than the modulus because of the generation of small primes.

## 4.2 RC RSA-II

RC RSA-II is another scheme based on reducing the computational overhead and memory consumption. Dual RSA Small-d [61] is having low cost in decryption side, but high cost in encryption side. In RC RSA-II, encryption cost of Dual RSA Small-d is tried to be reduced by shifting the online calculation of exponentiation to offline calculation. For this purpose, the features of Dual RSA Small-d [61] and DRSA [102] are combined to design RC RSA-II. Key generation method is same as Dual RSA Small-d:

### 4.2.1 Key Generation Method

In the algorithm, $n_e$=number of bits in $e$, $n_d$=number of bits in $d$ ($n_d < n/2$), $n$ =number of bits in moduli ($N_1$ and $N_2$), $n_k$=number of bits in security parameters ($k_1$ and $k_2$), $n/2$=number of bits in primes ($p_1, q_1$ and $p_2, q_2$):

1. Select the private exponent $d$ with $n_d$ bit.
2. Prime factors $p_1, q_1, p_2$ are computed as $p_1 = u_1u_2 + 1, p_2 = u_1v_2 + 1, q_1 = v_1v_2 + 1$ such that $u_1, v_1$ with $n_d$ bit and $u_2, v_2$ with $(n/2 - n_d)$ bit are random numbers satisfying $GCD(d, u_1u_2) = 1$ and $GCD(d, v_1v_2) = 1$.
3. Parameters $e$ and $k_1$ are computed such that $ed = 1 + k_1(p_1 - 1)(q_1 - 1)$.
4. Compute $q_2 = k_1u_2 + 1$, if $q_2$ is not prime then repeat the process by taking another random choice for the calculation of the prime factor $q_1$ in step 2.

The output is: $N_1 = p_1q_1$ and $N_2 = p_2q_2$

### 4.2.2 Encryption and Decryption Method

As the public exponent is of the order of the modulus, the encryption process is very time consuming. Here the approach is used to accomplish the costly exponentiation part before the real encryption; the only computation of multiplication is performed online.

**Offline Encryption**

Following statements can be calculated offline (before encrypting the message).

1. Select $R$ as any random number $R \in \mathbb{Z}^*_{N_1}$
2. Calculate $C' = (R - 1)^e \mod N_1$
3. Calculate $R^{-1} \mod N_1$

**Encryption Method**

To encrypt any plain text $M$,

$$C = M * R^{-1} \mod N_1 \tag{4.38}$$

$(C', C)$ is the required cipher text.

**Decryption Method**

To decrypt the cipher text $(C', C)$, the receiver has to follow the below mentioned steps:

1. Calculate $R = C'^d \mod N_1 + 1$
2. Calculate the message $M = R * C \mod N_1$

Extra cost of multiplication in decryption method is negligible. In Dual RSA Small-d [61], encryption exponent ($e$) is taken to be very large; hence the efficiency of encryption process becomes very slow. In RC RSA-II, since the pairs like $((R-1)^e \mod N_1, R^{-1} \mod N_1)$ can be computed well in advance (offline calculation), the online encryption requires only one multiplication modulo $N_1$. This shifting of calculation (from online to offline) results in very fast online encryption. Due to only one extra multiplication modulo $N_1$, decryption side is computationally as expensive as Dual RSA Small-d.

### 4.2.3 Security Analysis

As the encryption exponent is considered to be of the order of the modulus, small-e attack is not applicable in this variant. Due to the use of small decryption exponent, small-d attack is critical. Introduction of random parameter makes the discussion of semantic security worth. The known attacks due to small-d are studied and presented here for the security of RC RSA-II. In this analysis, the scheme is considered to be secure from side channel attack. The scheme can further be analyzed against partial key exposure attack by exposing the variant to different sources of side channel attack.

**Small-d Attack**

In RC RSA-II, parameters remain same as in Dual RSA Small-d, thus security constraints also remain unchanged. The best known attack on Small-d is due to Boneh and Durfee [36]; i.e. $d < N^{0.292}$.

Due to the two RSA equations involved, to avoid certain attacks following constraints have to be satisfied.

Reconsidering equation (4.20) $k_1N_1 - k_lN_2 + k_lt_2 + k_mt_2 - k_1t_1 - k_mN_2 = 0$,

Roots of this polynomial are $x_0, y_0, z_0$; where $x = k_1$, $y = k_l$, $z = k_lt_2 + k_mt_2 - k_1t_1$ and $C = k_mN_2$ (C is constant). The polynomial can be solved for the upper bounds $X \geq x_0$, $Y \geq y_0$, $Z \geq z_0$, provided that N is very large and $XYZ < M$, where $M = ||f(xX, yY, zZ)||_\infty = 2^{n_e+n_d}$. Bounds X, Y, Z are as follows:

$X = 2^{n_e+n_d-n}$, $Y = 2^{n_e+n_d-n-m}$ and $Z = 2^{n_e+n_d-n/2}$

This polynomial can be solved for $XYZ < M$ if:

$n_e + n_d < 5n/4 + m/2$ or $n_d < n/4 + m/2$, which is the same as in Dual RSA small-d [61]. The other bound given in [61] is: $n_d > n/3$

Thus the security constraints for RC RSA-II are:

$$n_d > 0.292 \tag{4.39}$$
$$n_d > n/4 + m/2 \tag{4.40}$$
$$n_d > n/3 \tag{4.41}$$

Last constraint is the most effective one; i.e. RC RSA-II is not secure for $n_d < 341$ (for moduli=1024 bits) and $n_d < 682$ (for moduli=2048 bits)

The study regarding Dual RSA [61] in Hinek [44] and Sarkar and Maitra [93] proved that any other attack does not apply to Dual RSA Small-d. Hence RC RSA-II is secure with the above mentioned bound for $n_d$.

### 4.2.4 Complexity Analysis

Complexity of the key generation method is dependent on the prime number generation. Complexity involves the computations of random prime numbers and

testing for primality. The complexity for random prime number is $O(n)$ and for primality testing it is $O(n_p^3)$ ($n_p = n/2$, where $n$ is the size of the modulus).

### Encryption Method

RSA encryption is done using exponentiation operation. In this scheme, the computation involved in the exponentiation is done well in advance (when the system is relatively free), thus the online computations are free from exponentiation operation. RC RSA-II is only computing the modular multiplication, thus the encryption complexity is:

$$Enc(C) = n^2 \tag{4.42}$$

### Decryption Method

In the decryption method, the exponentiation cost is involved using square and multiply method. For $n_d$-bit decryption exponent, the decryption complexity of RC RSA-II is;

$$Dec(C) = (n_d + n_d/2)n^2 \tag{4.43}$$

For $n_d = 341$ and $n = 1024$;

$$Dec(C) = 511n^2 \tag{4.44}$$

### Complexity in parallelization

In RC RSA-II, as the decryption process cannot be divided into smaller processes, the parallelization of the process does not contribute in enhancement of the efficiency.

### Memory Usage

Due to the use of random number $R$, the memory consumption by RC RSA-II seems to have increased. This effect can defeat the purpose of using the scheme in resource constrained environments (with less memory).

For making the cryptosystem secure against chosen plaintext message attack, some randomization is needed which results in message expansion. In RSA also, the padding scheme [16] is used which increases the length of the actual message. RSA with padding scheme [16] gets the message expansion of 2.3 [102]. Because of the use of n bit random number, RC RSA-II gets the message expansion of 2 for moduli of size 1024 bits [102]. Thus RC RSA-II has better message expansion as

compared to RSA with padding scheme. This factor is ignored in the computaions for the sake of maintaining the simplicity.

Like RC RSA-I, RC RSA-II is also based on two instances of RSA with common public and private exponent. Memory consumption is calculated by three parameters: modulus ($N$), public exponent ($e$) and decryption exponent ($d$). Thus, the memory consumed by two RSA instances with common $e$ and $d$ is; $2n$ bits for two modulus ($N_1$ and $N_2$), $n$ bits for encryption exponent $e$ and $n/3$ bits for the decryption exponent ($d$). Memory usage by RC RSA-II is:

$$Mem(C) = (2n + n + n/3) = 3.333bits \tag{4.45}$$

Hence the complexity is:

$$\begin{aligned} Enc(C) &= n^2 \\ Dec(C) &= 511n^2 \\ Mem(C) &= (3.333n)bits \end{aligned}$$

### 4.2.5 Implementation Details

RC RSA-II is implemented in C++ on windows operating system. To deal with big modulus size, the scheme is implemented using NTL with GMP (section 4.1.5 and Appendix A).

Key generation, encryption and decryption methods are implemented and run for 100 times to take the average values. Following results were found for $n = 1024$ and $n = 2048$ bits. Details of the examples are given in appendix B.3.

With modulus size 1024 bits:

Size($N_1$)=1024 bits

Size($N_2$)=1024 bits

Size(d)=342 bits

Size(e)=1024 bits

Time consumed by key generation=367.092 ms

Time consumed by encryption=0.031 ms

Time consumed by decryption=1.41 ms

As the value of encryption exponent is very high of the order of the modulus, encryption time is expected to be very high. But completing the calculation of exponentiation before the actual encryption, the online time consumption comes out to be very less.

TABLE 4.3: Implementation Results of RC RSA-II for N=1024bits

| $n_d$ | Random Number Generated | Key Generation Time (ms) |
|---|---|---|
| 342 | 1135 | 367.092 |
| 380 | 1217 | 347.806 |
| 400 | 1258 | 363.978 |
| 420 | 1117 | 324.571 |

The algorithm is also implemented with modulus size 2048 bits and the following values are recorded:

bits in $N_1$=2048

bits in $N_2$=2046

bits in $d$=684

bits in $e$=2047

TABLE 4.4: Implementation Results of RC RSA-II for N=2048 bits

| $n_e$ | Random Number Generated | Key Generation Time (ms) |
|---|---|---|
| 684 | 2486 | 4645.7 |
| 760 | 2362 | 5613.73 |
| 800 | 2306 | 5108.14 |
| 840 | 2374 | 4540.63 |

Time taken by Key generation method=4645.7 ms

Time taken by encryption method=0.109 ms

Time taken by decryption method=8.42 ms

In this result also the encryption cost is very less as with the modulus size 1024 bits.

Efficiency of the algorithm depends on the size of the modulus. From table (4.3 and 4.4), random number generated are of the order of the size of the moduli (1024 bits in table 4.3 and 2048 bits in table 4.4).

## 4.3 RC RSA-III

RC RSA-III is the improvement over third scheme of Dual RSA, i.e Dual Generalized Rebalanced RSA (DGRR) [61]. Besides less memory consumption for two RSA instances, work is done to make DGRR more suitable for resource constrained environments. In RC RSA-III, key generation method of DGRR is extended by using multiple prime modulus. For $x$ primes, $x$ small CRT exponents are used for the computation. Thus the prime factors and CRT exponents with small bit size make the computational performance of the proposed scheme better.

As the basic RSA key equation is $ed = 1 + k\phi(N)$; due to the use of multiple primes in RC RSA-III, the key equation becomes:

$$ed_i = 1 + k_{p_i}(p_i - 1) \tag{4.46}$$

$$ed_i = 1 + k_{q_i}(q_i - 1) \tag{4.47}$$

### 4.3.1 Key Generation Method

The algorithm is described for $x$ primes; $n_e + n_d = n_k + n/x$ and $n_e < n/x$. Here $n_e$=number of bits in $e$, $n_d$=number of bits in $d_i$, $n$ =number of bits in moduli ($N_1$ and $N_2$), $n_k$=number of bits in security parameters ($k_{p_i}$ and $k_{q_i}$), $n/x$=number of bits in primes $p_1, p_2, ..., p_x$ and $q_1, q_2, ..., q_x$.

1. Select a random integer $e$ with size $n_e$ - bit and calculate integer $k$ as $(n/x - n_e)/n_k$.

2. Select $(k-1)$ random integers with size $n_k$-bit integers $p_{a_1}, p_{a_2}, ..., p_{a_{(k-1)}}$ and $k^{th}$ even integer $p_{a_k}$ such that the product of all the terms $p_{a_1}$ to $p_{a_k}$ is $p_a$ with size $(n/x - n_e)$ and $GCD(e, p_a) = 1$.

3. Select a random integer $k_{p_1}$ with size $n_k$ bit such that $GCD(e, k_{p_1}) = 1$.

4. Parameters $d_i$ and $p_b$ are so computed as to satisfy the CRT key equation $ed_i = k_{p_1} p_a p_b + 1$.

5. Calculate $p_1 = p_a p_b + 1$, if it is not prime then repeat the process for other choice of $k_{p_1}$ in step 3.

6. Calculate $p_2 = (k_{p_1} p_a p_b / p_{a_{i'}}) + 1$ for any value of $1 \leqslant i' \leqslant (k-1)$, if $p_2$ is not prime for any value of $i'$, then goto step 3.

7. Step 2-6 are repeated for $x$ prime pairs, i.e. $(p_1, p_2), (q_1, q_2), ...$ and so on upto $x$ pairs.

For three prime RC RSA-III, $N_1 = p_1 q_1 r_1, N_2 = p_2 q_2 r_2, k_{p_2} = p_{a_{i'}}, k_{q_2} = q_{a_{j'}}, k_{r_2} = r_{a_{l'}}$ and so on, where $i', j', l'...$ are the number index in the array $p_a, q_a, r_a....$

In the algorithm:

$p_1 = p_a p_b + 1$, $p_2 = p'_a p_b + 1$

$q_1 = q_a q_b + 1$, $q_2 = q'_a q_b + 1$

$r_1 = r_a r_b + 1$, $r_2 = r'_a r_b + 1$

For,

$p_a = p_{a_1} p_{a_2} ... p_{a_k}$, $p'_a = p_a k_{p_1} / p_{a_{i'}}$

$q_a = q_{a_1} q_{a_2} ... q_{a_k}$, $q'_a = q_a k_{q_1} / q_{a_{j'}}$

$r_a = r_{a_1} r_{a_2} ... r_{a_k}$, $r'_a = r_a k_{r_1} / r_{a_{l'}}$

**Correctness of Key Generation Method**

The correctness of the algorithm can be proved as below:

$$\begin{aligned} ed_i &= 1 + k_{p_1}(p_1 - 1) && (4.48)\\ &= 1 + k_{p_1}(p_a p_b) && (4.49)\\ &= 1 + k_{p_1}(p_{a_1} ... p_{a_{i'}} ... p_{a_k} p_b) && (4.50)\\ &= 1 + p_{a_{i'}}(p_{a_1} ... k_{p_1} ... p_{a_k} p_b) && (4.51)\\ &= 1 + k_{p_2}(p_2 - 1) && (4.52) \end{aligned}$$

Hence the equation used in RC RSA-III is a valid RSA CRT key equation(eq. 2.36).

### 4.3.2 Encryption and Decryption Method

Encryption is done in the same way as in standard RSA method. For two RSA instances with moduli ($N_1$ or $N_2$), cipher text ($C_1$ or $C_2$) is calculated by the exponentiation of the message ($M_1$ or $M_2$):

$$C_1 = M_1^e \mod N_1 \tag{4.53}$$

or

$$C_2 = M_2^e \mod N_2 \tag{4.54}$$

(where $e$ is the common public exponent for two RSA exponents)

For decryption method, message ($M_1$ and $M_2$) can be computed by exponentiating the cipher text by small decryption exponents (called CRT exponents) $d_1, d_2, ..., d_x$.

For $1 \leq i \leq x$:

$$M_{1_i} = C_i^{d_i} \mod p_{1_i} \tag{4.55}$$

where $C_i = C_1 \mod p_{1_i}$ and

$$M_{2_i} = C_i^{d_i} \mod p_{2_i} \tag{4.56}$$

where $C_i = C_2 \mod p_{2_i}$

($p_{1_i} = p_1, q_1, r_1$ and so on; $p_{2_i} = p_2, q_2, r_2$ and so on) $M_1$ and $M_2$ can be computed by using CRT (subsection 2.1.6).

### 4.3.3 Security Analysis

MultiPrime security is studied in [44], [72] and [74], almost all the described attacks worked on the RSA key equation $ed = 1 + k\phi(N)$ with multiple primes. Proposed scheme is based on using CRT key equation, $ed_i = 1 + k_i(p_i - 1)$; all the attacks on MultiPrime RSA become ineffective or less effective with CRT components. Various known attacks are studied and analyzed in terms of RC RSA-III with

three primes. As the scheme is designed with multiple primes, factorization attack is discussed for the valid number of primes used for the modulus. Besides this, CRT attack and parameter selection attacks (with known and unknown values of the security parameters) are critical for RC RSA-III. Like RC RSA-I and RC RSA-II, RC RSA-III is also assumed to be free from side channel attack. The scheme can further be studied against the environment prone to side channel attack by which few bits of the unknown parameters can be made known (partial key exposure attack).

**Factorization Attack**

This attack is based on two popular factoring methods: NFS (general purpose factoring method) and ECM (special purpose factoring method). The estimated maximum number of balanced primes that are considered to be secure are 3, 3, 4, 5 for 1024, 2048, 4096 and 8192 bit modulus respectively [71]. In RC RSA-III, modulus $N$ (with size 1024 bits or 2048 bits) is considered to be the product of three factors. Hence, the proposed scheme is considered to be secure against factoring attack of both general purpose method due to large modulus size and special purpose method due to large size of the prime factors.

**Small CRT Attack**

For cryptanalysis of RSA CRT, small CRT exponents can be attacked with complexity $O(min(\sqrt{d_p}, \sqrt{d_q}))$ [63]. The same is applicable for RC RSA-III, hence $n_d > 2m$ (where $m$ is the security parameter below which exhaustive search is possible).

In 2007, Jochemsz and May [64] described the cryptanalytic attack on RSA CRT, according to which the modulus can be factored in polynomial time if the CRT-exponents $|d_p| < 74$ bits for modulus size 1024 bits. For multi-prime RSA, this bound will be much smaller due to smaller size of the parameters. It will be so much smaller that an exhaustive search for CRT-exponents will be feasible. Hence this attack will not contribute much in cryptanalysis of RC RSA-III.

**Unknown $k_1$ and $k_2$ for $n_e + n_d < 7n/18 + m/3 - 3/2$ (continued fraction attack)**

As $N_1 = p_1q_1r_1$ and $N_2 = p_2q_2r_2$: For some $(i', j', m') \in \{1, k-1\}$, where $k = [(n/3 - n_e)/n_k]$

$|p_{a_i}| = |q_{a_i}| = |r_{a_i}| = |k_{p_1}| = |k_{q_1}| = |k_{r_1}| = n_k,\ i \in \{1, k-1\}$

$|p_a| = |q_a| = |r_a| = [n/3 - n_e]$

$|p_b| = |q_b| = |r_b| = [n_e]$

$|d_p| = |d_q| = |d_r| = [n/3 - n_e + n_k]$

As $N_1 = p_1 q_1 r_1$:

$$N_1 = (p_a p_b + 1)(q_a q_b + 1)(r_a r_b + 1) \tag{4.57}$$

$$N_1 - 1 = p_a p_b q_a q_b r_a r_b + p_a p_b q_a q_b + p_a p_b r_a r_b + q_a q_b r_a r_b + p_a p_b + q_a q_b + r_a r_b \tag{4.58}$$

Dividing both the sides by $p_{a'} p_b q_{a'} q_b r_{a'} r_b$:

$$\frac{N_1 - 1}{p_{a'} p_b q_{a'} q_b r_{a'} r_b} = \frac{p_{a'_i} q_{a'_j} r_{a'_m}}{k_{p_1} k_{q_1} k_{r_1}} + \frac{p_a p_b q_a q_b + p_a p_b r_a r_b + q_a q_b r_a r_b + p_a p_b + q_a q_b + r_a r_b}{p_{a'} p_b q_{a'} q_b r_{a'} r_b} \tag{4.59}$$

$$\frac{N_1 - 1}{p_{a'} p_b q_{a'} q_b r_{a'} r_b} < \frac{p_{a'_i} q_{a'_j} r_{a'_m}}{k_{p_1} k_{q_1} k_{r_1}} + \frac{6(2^{n/3})^2}{(\frac{1}{3} 2^{n/3})^3} \tag{4.60}$$

From equation: $N_2 = p_2 q_2 r_2$

$$N_2 = (p_{a'} p_{b'} + 1)(q_{a'} q_{b'} + 1)(r_{a'} r_{b'} + 1) \tag{4.61}$$

Dividing both the sides by $p_{a'} p_b q_{a'} q_b r_{a'} r_b$:

$$\frac{N_2 - 1}{p_{a'} p_b q_{a'} q_b r_{a'} r_b} = 1 + \frac{p_{a'} p_b q_{a'} q_b + p_{a'} p_b r_{a'} r_b + q_{a'} q_b r_{a'} r_b + p_{a'} p_b + q_{a'} q_b + r_{a'} r_b}{p_{a'} p_b q_{a'} q_b r_{a'} r_b} \tag{4.62}$$

$$\frac{N_2 - 1}{p_{a'} p_b q_{a'} q_b r_{a'} r_b} > 1 \tag{4.63}$$

Dividing equation (4.60) by (4.62):-

$$\frac{N_1 - 1}{N_2 - 1} - \frac{p_{a'_i} q_{a'_j} r_{a'_m}}{k_{p_1} k_{q_1} k_{r_1}} < \frac{6 * 27}{2^{n/3}} < \frac{1}{2^{n/3-8}} \tag{4.64}$$

$$< \frac{1}{2(k_{p_1} k_{q_1} k_{r_1})^2} \tag{4.65}$$

$$2(k_{p_1} k_{q_1} k_{r_1})^2 < 2^{n/3-8} \tag{4.66}$$

$$3n_k < \frac{n}{6} - \frac{9}{2} \tag{4.67}$$

By adding a few bits ($m$ bits) of exhaustive search in the bound:

$$n_k < \frac{n}{18} + \frac{m}{3} - \frac{3}{2} \tag{4.68}$$

Or Using $n_e + n_d = n_k + n/3$:

$$n_e + n_d < \frac{7n}{18} + \frac{m}{3} - \frac{3}{2} \tag{4.69}$$

Thus, if the inequalities (4.68) or (4.69) holds, one can find the value of $\frac{k_{p_2} k_{q_2} k_{r_2}}{k_{p_1} k_{q_1} k_{r_1}}$ or $\frac{k_{2'}}{k_{1'}}$

**Unknown $k_1$ and $k_2$ for $19n_d + 10n_e < 16n/3 - \psi$ (Lattice attack)**

For $e = N^{\alpha}$; $d_p, d_q, d_r < N^{\delta}$; $g = N^{\gamma}, g_1 = N^{\gamma_1}, g_2 = N^{\gamma_2}$; where $g = GCD(N_1 - 1, N_2 - 1)$, $g_1 = GCD(e, (N_1 - 1)/g), g_2 = GCD(e, (N_2 - 1)/g), N_1' = (N_1 - 1)/gg_1, N_2' = (N_2 - 1)/gg_2$.

Key equations (from eq. 4.46) for first instance:

$$ed_p = k_{p_1}(p_1 - 1) + 1 \tag{4.70}$$
$$ed_q = k_{q_1}(q_1 - 1) + 1 \tag{4.71}$$
$$ed_r = k_{r_1}(r_1 - 1) + 1 \tag{4.72}$$

Multiplying these three equations:

$$\begin{aligned} &e^3 d_p d_q d_r + e^2[d_p d_r(k_{q_1} - 1) + d_q d_r(k_{p_1} - 1) + d_p d_q(k_{r_1} - 1)] + e[d_p(k_{q_1} k_{r_1} \\ &\quad - k_{r_1} - k_{q_1} + 1) + d_q(k_{p_1} k_{r_1} - k_{r_1} - k_{p_1} + 1) + d_r(k_{p_1} k_{q_1} - k_{p_1} - k_{q_1} + 1)] \\ &\quad - k_{p_1} k_{q_1} - k_{q_1} k_{r_1} - k_{p_1} k_{r_1} + k_{p_1} + k_{q_1} + k_{r_1} - 1 = k_{p_1} k_{q_1} k_{r_1}(N_1 - 1) \end{aligned} \tag{4.73}$$

Key equations (from eq. 4.47) for second instance:

$$ed_p = k_{p_2}(p_2 - 1) + 1 \tag{4.74}$$
$$ed_q = k_{q_2}(q_2 - 1) + 1 \tag{4.75}$$
$$ed_r = k_{r_2}(r_2 - 1) + 1 \tag{4.76}$$

Multiplying these three equations:

$$\begin{aligned} &e^3 d_p d_q d_r + e^2[d_p d_r(k_{q_2} - 1) + d_q d_r(k_{p_2} - 1) + d_p d_q(k_{r_2} - 1)] + e[d_p(k_{q_2} k_{r_2} \\ &\quad - k_{r_2} - k_{q_2} + 1) + d_q(k_{p_2} k_{r_2} - k_{r_2} - k_{p_2} + 1) + d_r(k_{p_2} k_{q_2} - k_{p_2} - k_{q_2} + 1)] \\ &\quad - k_{p_2} k_{q_2} - k_{q_2} k_{r_2} - k_{p_2} k_{r_2} + k_{p_2} + k_{q_2} + k_{r_2} - 1 = k_{p_2} k_{q_2} k_{r_2}(N_2 - 1) \end{aligned} \tag{4.77}$$

Combining equations (4.73) and (4.77), we get

$$f(x_1, x_2, x_3, x_4, x_5, x_6, x_7) = 0 \mod N_1' N_2' \tag{4.78}$$

Using Lattice method for linear polynomial, the roots of the above function can be solved if

$X_1 X_2 X_3 X_4 X_5 X_6 X_7 < N_1' N_2'$

Where $N_1' N_2' = (N_1 - 1)(N_2 - 1)/g^2 g_1 g_2$ and $X_1, X_2, X_3, X_4, X_5, X_6, X_7$ are the upper bounds for $x_1, x_2, x_3, x_4, x_5, x_6, x_7$ with the below mentioned values:

$x_1 = d_p d_q d_r$

$x_2 = d_p d_r(k_{q_1} - 1) + d_q d_r(k_{p_1} - 1) + d_p d_q(k_{r_1} - 1)$

$x_3 = d_p(k_{q_1} k_{r_1} - k_{r_1} - k_{q_1} + 1) + d_q(k_{p_1} k_{r_1} - k_{r_1} - k_{p_1} + 1) + d_r(k_{p_1} k_{q_1} - k_{p_1} - k_{q_1} + 1)$

$x_4 = k_{p_1} + k_{q_1} + k_{r_1} - k_{p_1} k_{q_1} - k_{q_1} k_{r_1} - k_{p_1} k_{r_1} - 1$

$x_5 = d_p d_r(k_{q_2} - 1) + d_q d_r(k_{p_2} - 1) + d_p d_q(k_{r_2} - 1)$

$x_6 = d_p(k_{q_2} k_{r_2} - k_{r_2} - k_{q_2} + 1) + d_q(k_{p_2} k_{r_2} - k_{r_2} - k_{p_2} + 1) + d_r(k_{p_2} k_{q_2} - k_{p_2} - k_{q_2} + 1)$

$x_7 = k_{p_2} + k_{q_2} + k_{r_2} - k_{p_2} k_{q_2} - k_{q_2} k_{r_2} - k_{p_2} k_{r_2} - 1$

The upper bounds are defined as:

$X_1 = 2^{3n_d}$

$X_2 = 2^{3n_d + n_e - n/3}$

$X_3 = 2^{3n_d + 2n_e - 2n/3}$

$X_4 = 2^{2n_d + 2n_e - 2n/3}$

$X_5 = 2^{3n_d + n_e - n/3}$

$X_6 = 2^{3n_d + 2n_e - 2n/3}$

$X_7 = 2^{2n_d + 2n_e - 2n/3}$

The modular polynomial can be solved for the following bound:

$$19n_d + 10n_e < \frac{16n}{3} - \psi \tag{4.79}$$

or

$$n_k < \frac{9n_e - n}{19} - \psi \tag{4.80}$$

(where $\psi = bits(g^2 g_1 g_2)$, which is considered to be very small)

With the bound of (4.79), the following parameters can be obtained without any relevant information:

$k_{p_1}+k_{q_1}+k_{r_1}-k_{p_1}k_{q_1}-k_{q_1}k_{r_1}-k_{p_1}k_{r_1}-1$ and $k_{p_2}+k_{q_2}+k_{r_2}-k_{p_2}k_{q_2}-k_{q_2}k_{r_2}-k_{p_2}k_{r_2}-1$

Computing the equation (4.73) mod $e$:

$$\left(\frac{N_1 - 1}{gg_1}\right) gg_1 k_{p_1} k_{q_1} k_{r_1} = (k_{p_1} + k_{q_1} + k_{r_1} - k_{p_1}k_{q_1} - k_{q_1}k_{r_1} - k_{p_1}k_{r_1} - 1) \mod e \tag{4.81}$$

or

$$gg_1 k_{p_1} k_{q_1} k_{r_1} < e \tag{4.82}$$

or

$$n_k < \frac{n_e - \tau}{3} \tag{4.83}$$

(where $\tau$ is the number of bits in the term $gg_1$) or

$$n_k < \frac{n_e}{3} - \epsilon \tag{4.84}$$

For this relation $\epsilon$ is considered as very small. From the above inequality, $k_1 = k_{p_1}k_{q_1}k_{r_1}$ and $k_2 = k_{p_2}k_{q_2}k_{r_2}$ can be calculated.

Thus, even with the combination of both the bounds given in (4.79) and (4.84), the parameters $k_{p_1}, k_{q_1}, k_{r_1}$ can not be calculated.

**Unknown $k_1$ and $k_2$ for $n_d < \frac{n}{3} - \frac{8n_e}{11}$ (Lattice attack)**

Subtracting equation (4.77) from equation (4.73), we can get

$$\begin{aligned} e^2[d_p d_r(k_{q_1}-1)+d_q d_r(k_{p_1}-1)+d_p d_q(k_{r_1}-1)-d_p d_r(k_{q_2}-1)-d_q d_r(k_{p_2}-1)-d_p d_q(k_{r_2}- \\ 1)]+e[d_p(k_{q_1}k_{r_1}-k_{r_1}-k_{q_1}+1)+d_q(k_{p_1}k_{r_1}-k_{r_1}-k_{p_1}+1)+d_r(k_{p_1}k_{q_1}-k_{p_1}-k_{q_1}+1)-d_p(k_{q_2} \\ k_{r_2}-k_{r_2}-k_{q_2}+1)-d_q(k_{p_2}k_{r_2}-k_{r_2}-k_{p_2}+1)-d_r(k_{p_2}k_{q_2}-k_{p_2}-k_{q_2}+1)]-k_{p_1}k_{q_1}-k_{q_1}k_{r_1}-k_{p_1} \\ k_{r_1} + k_{p_1} + k_{q_1} + k_{r_1} + k_{p_2}k_{q_2} + k_{q_2}k_{r_2} + k_{p_2}k_{r_2} - k_{p_2} - k_{q_2} - k_{r_2} + k_{p_2}k_{q_2}k_{r_2}(N_2 - 1) \\ = k_{p_1}k_{q_1}k_{r_1}(N_1 - 1) \end{aligned} \tag{4.85}$$

The above equation can be treated as modulo $(N_1 - 1)$ getting the following polynomial,

$$f(x, y, z, w) = e^2x + ey + z + (N_2 - 1)w \tag{4.86}$$

with upper bound, X,Y,Z,W:

$X = 2^{n_e+3n_d-n/3}$

$Y = 2^{2n_e+3n_d-2/3n}$

$Z = 2^{2n_e+2n_d-2/3n}$

$W = 2^{3n_e+3n_d-n}$

The following inequality will solve the roots of equation (4.85), for $XYZW < N_1 - 1$, this solves to be

$$n_d < \frac{n}{3} - \frac{8n_e}{11} \tag{4.87}$$

or

$$n_k < \frac{3n_e}{11} \tag{4.88}$$

With the above bound, one can solve the value for $k_2 = k_{p_2}k_{q_2}k_{r_2}$, similarly $k_1 = k_{p_1}k_{q_1}k_{r_1}$ can be solved.

**Known $k_1$ and $k_2$**

Considering equation (4.26):

$k_1'(N_1 - t_1) = k_2'(N_2 - t_2)$

and equation (4.29):

$f_{N_1}(x, y) = k_1'k_2'x - k_2'y + k_2'N_2 + k_1'C_1$

Here$|k_1'| = |k_2'| = 3n_e + 3n_d - n - \gamma$

According to these values the bound becomes

$$n_e + n_d > 4n/9 + (\gamma)/3 \tag{4.89}$$

The polynomial can be solved for the above bound. Hence for known value of $k_1$ and $k_2$, the RSA equation can be solved probabilistically.

Hence, with the following inequalities, $k_1 = k_{p_1}k_{q_1}k_{r_1}$ and $k_2 = k_{p_2}k_{q_2}k_{r_2}$ can be computed, but $k_{p_1}, k_{q_1}, k_{r_1}, k_{p_2}, k_{q_2}, k_{r_2}$ cannot be computed directly. These parameters can only be computed by factorizing $k_1$ and $k_2$.

- $n_e + n_d < \frac{7n}{18} + \frac{m}{3} - \frac{3}{2}$ or $n_k < \frac{n}{18} + \frac{m}{3} - \frac{1}{2}$
- $2n_e + 3n_d < n - \epsilon$ or $n_k < \frac{n_e}{3} - \epsilon$
- $24n_e + 33n_d < 11n$ or $n_k < \frac{3n_e}{11}$
- $11n_d + 5n_e < \frac{8}{3}n + m$ or $n_k < \frac{6n_e - n + m}{11}$

With known $k_1$ and $k_2$, the bound $n_e + n_d < 4n/9 + (\gamma)/3$ must be satisfied to avoid the attacks.

**Security Summary of RC RSA-III**

As discussed above, the security can be at risk by knowing $k_1$ and $k_2$ but with very high complexity.

For $x$ prime modulus and known $k_1$:

$n_e < [\frac{x-1}{x^2}]n - m$

and due to small CRT attack:

$n_d \geq 2m$

also for known $k_1$ and $k_2$, $n_e + n_d < 4n/9 + (\gamma)/3$

For known $k_{p_i}$ parameters, $n_e < 147$ bits for modulus size 1024 bits and $n_e < 327$ bits for modulus size 2048 bits. CRT exponent $n_d$ varies according to the chosen value of $n_e$.

For unknown $k_{p_i}$, $n_k > 110$ bits for 1024 bit modulus and $n_k > 227$ bits for 2048 bit modulus. For small CRT exponents, $n_e$ needs to be taken as large values. As the size of CRT exponent cannot be less than 160 bits (for 1024 bit modulus) and 256 bits (for 2048 bit modulus), size of public exponent need not to be taken greater than 292 bits.

In the proposed scheme, the example taken is: $n_e = 140$ bits and $n_d = 219$ bits.

### 4.3.4 Complexity Analysis

As compared to Dual Generalized RSA [61], number of steps are increased in RC RSA-III. In the key generation algorithm, primality testing is the complex operation. Except this computation, all the steps are implemented using small computations of addition and multiplication. In the algorithm every pair must be simultaneously prime. Complexity of generating the random prime number is $O(n)$ and the complexity of primality testing is $O(n_p^3)$, where $n_p$ is the bit-size of the number to be tested. In RC RSA-III the prime generation becomes complex as prime numbers are generated according to the special structure required in the algorithm.

#### Encryption Method

Like Dual RSA, RC RSA-III is designed for two RSA instances ($N_1$ and $N_2$) and encryption is done by exponentiation operation using square and multiply method. For this scheme, $n_e = 140$ bits or $e = 2^{139} + 1$; only 139 modular square and one modular multiplication operations are performed. Using square and multiply method, the encryption complexity of RC RSA-III for n-bit modulus is:

$$Enc(C) = 140n^2 \tag{4.90}$$

#### Decryption Method

In RC RSA-III, RSA keys are generated for $k$ primes. In the decryption method $k$ exponentiation are performed using square and multiply method to get $x$ intermediate messages. For $n_e = 140$ bits; the size of the decryption exponent ($n_d$) must be 219 bits. The decryption complexity (ignoring CRT computations) for $x$ primes;

$$Dec(C) = x * (n_d + 1/2n_d)n_p^2 \tag{4.91}$$

For $x = 3$, for $n_d = 219$ bits and $n_p = n/3$;

$$Dec(C) = 109n^2 \tag{4.92}$$

#### Complexity in Parallelization

Performance of encryption process does not gain any benefit, but the intermediate processes in decryption method can be implemented in parallel for the performance

gain. The decryption complexity in case of parallelization becomes:

$$Dec(C_p) = (n_d + n_d/2)n_p^2 \quad (4.93)$$

$$Dec(C_p) = 37n^2 \quad (4.94)$$

**Memory Usage**

Like previous schemes (RC RSA-I and RC RSA-II), RC RSA-III is also based on the design of RSA scheme for two RSA instances. Both the instances are having different moduli ($N_1$ and $N_2$) but common public ($e$) and private exponents ($d$). The memory consumed for two RSA instances using RC RSA-III is; 2n bits for two modulus ($N_1$ and $N_2$), 140 bits for encryption exponent ($e$) and 219 bits for the decryption exponent ($d$). Memory usage by RC RSA-III is;

$$Mem(C) = (2n + 359)bits \quad (4.95)$$

Complexity in RC RSA-III is:

$$Enc(C) = 140n^2$$

$$Dec(C) = 109n^2$$

$$Mem(C) = (2n + 359)bits$$

### 4.3.5 Implementation Details

RC RSA-III is implemented in C++ on windows operating system. The scheme is implemented with modulus size 1024 bits and 2048 bits. NTL with GMP (section 4.1.5, Appendix A) is used to deal with big modulus size.

Table 4.5 shows the time taken by key generation, encryption and decryption methods for RC RSA-III for modulus size 1024 bits. Results are taken for the implementation with different values of $n_e$ and $n_d$. Details of the examples are given in appendix B.4.

Following values are taken for the implementation of RC RSA-III with modulus size 1024. In the following example, encryption speed is fast and decryption speed is less:

bits in $N_1$=1024

bits in $N_2$=1022

bits in $p_1$=341

bits in $p_2 = 340$

bits in $q_1 = 342$

bits in $q_2$=341

bits in $r_1$=342

bits in $r_2$=342

bits in $e$=17

bits in $d_p$=340

bits in $d_q$=341

bits in $d_r$=342

bits in $k_{p_1}$=17

bits in $k_{q_1}$=17

bits in $k_{r_1}$=17

bits in $k_{p_2}$=17

bits in $k_{q_2}$=17

bits in $k_{r_2}$=17

Time consumed by key generation =218 ms

Time consumed by encryption =0.109 ms

Time consumed by decryption=0.63 ms

Thus the resultant performance shows less encryption time and more decryption time. In other example, the encryption and decryption processes are shown to be balanced:

bits in $N_1$=1023

bits in $N_2$=1024

bits in $p_1$=341

bits in $p_2$=342

bits in $q_1$=341

bits in $q_2$=342

bits in $r_1$=341

bits in $r_2$=342

bits in $e$=140

bits in $d_p$=219

bits in $d_q$=218

bits in $d_r$=219

bits in $k_{p_1}$=17

bits in $k_{q_1}$=17

bits in $k_{r_1}$=17

bits in $k_{p_2}$=17

bits in $k_{q_2}$=17

bits in $k_{r_2}$=17

Time consumed by key generation =405 ms

Time consumed by encryption=0.577 ms

Time consumed by decryption=0.47 ms

In this example the resultant performance is almost balanced for both the encryption and decryption methods.

Implementation results of RC RSA-III for modulus size 2048 bits are shown in table 4.6. Following values are taken as the result for the implementation of RC

TABLE 4.5: Implementation Results of RC RSA-III for N=1024 bits

| Public Exponent $n_e$ (bits) | Private Exponent $n_d$ (bits) | Random Number Generated | Key Generated Time (ms) |
|---|---|---|---|
| 140 | 218 | 7652 | 405.57 |
| 100 | 258 | 2218 | 314.109 |
| 50 | 308 | 1373 | 506.69 |
| 17 | 341 | 1887 | 415.328 |

RSA-III for modulus size 2048 bits. In the first example, values are shown for fast encryption speed and slow decryption speed:

bits in $N_1$=2047

bits in $N_2$=2048

bits in $p_1$=682

bits in $p_2$=682

bits in $q_1$=683

bits in $q_2$=683

bits in $r_1$=683

bits in $r_2$=684

bits in $e$=17

bits in $d_p$=682

bits in $d_q$=683

bits in $d_r$=683

bits in $k_{p_1}$=17

bits in $k_{q_1}$=17

bits in $k_{r_1}$=17

bits in $k_{p_2}$=17

bits in $k_{q_2}$=17

bits in $k_{r_2}$=17

Time consumed by key generation =1887 ms

Time consumed by encryption=0.328 ms

Time consumed by decryption=3.74 ms

Hence the time consumption is less for encryption and more for decryption.

TABLE 4.6: Implementation Results of RC RSA-III for N=2048 bits

| Public Exponent $n_e$ (bits) | Private Exponent $n_d$ (bits) | Random Number Generated | Key Generation Time (ms) |
|---|---|---|---|
| 17 | 683 | 17234 | 1887.78 |
| 100 | 600 | 17670 | 1218.56 |
| 200 | 500 | 15540 | 1405.34 |
| 300 | 400 | 20056 | 1373.67 |

The second example is the resultant value for balanced encryption and decryption operations for modulus size 2048 bits:

bits in $N_1$=2046

bits in $N_2$=2046

bits in $p_1$=682

bits in $p_2$=682

bits in $q_1$=683

bits in $q_2$=682

bits in $r_1$=683

bits in $r_2$=682

bits in $e$=300

bits in $d_p$=399

bits in $d_q$=399

bits in $d_r$=399

bits in $k_{p_1}$=17

bits in $k_{q_1}$=17

bits in $k_{r_1}$=17

bits in $k_{p_2}$=17

bits in $k_{q_2}$=17

bits in $k_{r_2}$=17

time consumed by key generation =1373 ms

time consumed by encryption=3.869 ms

time consumed by decryption =2.18 ms

This result gives almost approximate balanced performance for encryption and decryption speed.

The efficiency of key generation method can be discussed with random number generated to get the key pair. From table (4.5) and (4.6), random number generated are the multiple of modulus size. Random number generated are larger than the modulus size because of the strict condition of generating the prime factors in the method.

## 4.4 Comparative Analysis

In this section complexity of RSA variants is analyzed in terms of computational performance and memory requirement. The proposed schemes are improvement over Dual RSA. As the schemes are based on generation of two RSA instances, the variants which directly affects the performance of the proposed schemes are considered for comparison; the analysis include RSA (small-e)[4], RSA (small-d)[31], RSA CRT[54], MultiPrime RSA[56], Rebalanced RSA[31], Small-e Dual RSA[61], Small-d Dual RSA[61] and Dual Generalized Rebalanced RSA (DGRR)[61].

For the sake of simplicity, $N$ is considered to have 1024 bits in this section. Table (4.7) shows the comparison of computational performance and memory consumption of the proposed schemes with other RSA schemes. RSA cryptosystem uses

three parameters which consume memory; public exponent $n_e$, private exponent $n_d$ and the modulus $n$.

In table (4.7), for the calculation of encryption/decryption complexity, the values are approximated with parameter $n^2$ ($n = 1024$ where required) so that the variants can be better compared with one another. Comparison of RC RSA schemes with other variants is shown graphically in figure (4.1) and memory comparison is shown in figure (4.2).

TABLE 4.7: Complexity Comparison of the proposed schemes with existing RSA variants

| Sr. No. | RSA Variants | Public Exponent $n_e$ (bits) | Encryption Complexity | Private Exponent $n_d$ (bits) | Decryption Complexity | Memory Usage for two instances $2 * (n_e + n_d + n)$ |
|---|---|---|---|---|---|---|
| 1. | RSA Small-e[4] | 17 | $17n^2$ | 1024 | $1536n^2$ | 4n+34 |
| 2. | RSA Small-d[31] | 1024 | $1536n^2$ | 299 | $448n^2$ | 4.584n |
| 3. | RSA CRT[54] | 17 | $17n^2$ | 1024 | $384n^2$ | 4n+34 |
| 4. | MultiPrime RSA[56] | 17 | $17n^2$ | 341 | $170n^2$ | 4n+34 |
| 5. | Rebalanced RSA[31] | 1024 | $1536n^2$ | 160 | $120n^2$ | 4n+640 |
| 6. | Small-e Dual RSA[61] | 296 | $296n^2$ | 1024 | $1536n^2$ | 3.25n+40 |
| 7. | Small-d Dual RSA[61] | 1024 | $1536n^2$ | 341 | $511n^2$ | 3.333n |
| 8. | DGRR[61] | 140 | $140n^2$ | 390 | $292n^2$ | 2n+530 |
| 9. | RC RSA-I | 17 | $17n^2$ | 341 | $170n^2$ | 3n+17 |
| 10 | RC RSA-II | 1024 | $n^2$ | 341 | $511n^2$ | 3.333n |
| 11. | RC RSA-III | 140 | $140n^2$ | 219 | $109n^2$ | 2n+359 |

Efficiency of the variants (table 4.7) varies according to the bit-size of the exponents ($n_e$ and $n_d$). Bit-size of the public exponent in RC RSA-I is 17 bits whereas in Dual RSA Small-e the bit size is 296 bits; likewise private exponent in RC RSA-I is of 341 bits (1024 bits in Dual RSA Small-e). Small bit size of the exponent makes the variant RC RSA-I more efficient as compared to Dual RSA Small-e. In the same way, RC RSA-III is more efficient than DGRR in terms of decryption complexity; size of the private exponent in RC RSA-III is 219 bits whereas in DGRR the size is 390 bits. Also, in comparison to Dual RSA Small-e and DGRR, the small bit-size used in RC RSA-I and RC RSA-III makes the memory consumption low. RC RSA-II is using same bit-size as in Dual RSA Small-d, here

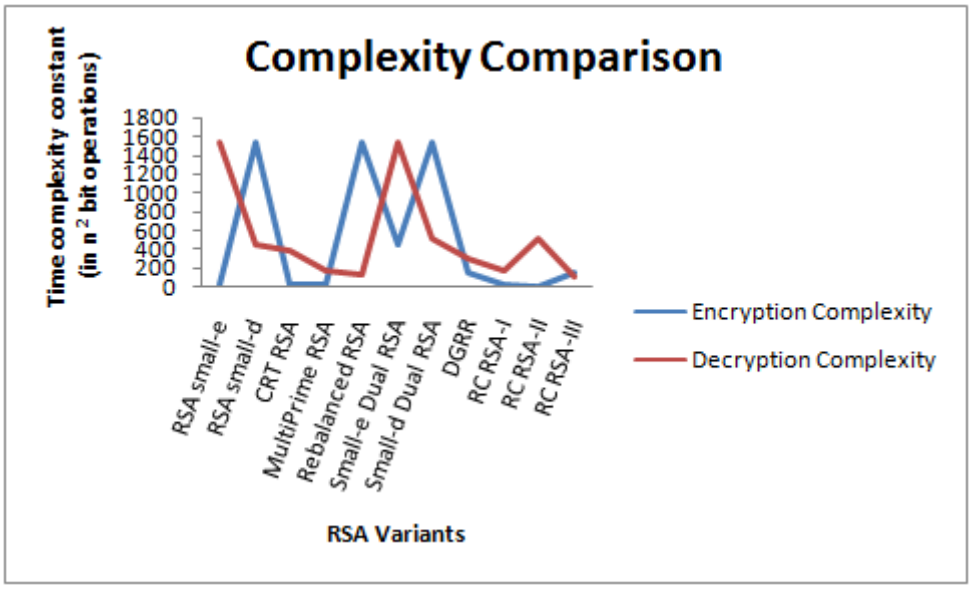


FIGURE 4.1: Comparison of RC RSA Schemes with existing RSA variants

the efficiency gain in encryption speed is not due to bit-size of the exponent. Here the offline work improves the efficiency of the scheme.

It is clear from table (4.7) that decryption speed of RC RSA-I is approximately 9 times faster than Dual RSA Small-e. The encryption cost becomes very low of the order of standard RSA; i.e. the encryption cost becomes 17 times faster than Dual RSA Small-e. Also, due to the use of small encryption exponent, memory usage by RC RSA-I reduces by almost $n/4$ bits. In RC RSA-II the encryption cost becomes almost negligible due to the computations completed before the real encryption. Decryption cost remains the same as in Dual RSA Small-d. As the parameters used in RC RSA-II remains the same, the memory usage is same as in Dual RSA Small-d. In RC RSA-III, the encryption and decryption complexity becomes less as compared to DGRR. For same encryption speed as in DGRR, the decryption speed becomes approximately 2.7 times faster than DGRR. Performance gain can be shifted towards encryption or decryption sides according to the requirement. Due to the use of small exponents ($e$ or $d$), memory consumption reduces by approximately 170 bits.

As the variants are implemented on single processing environment, the complexity of the variants in table (4.7) is calculated according to that. The efficiency (in terms of decryption complexity) of the variants will further be improved if parallel processing environment is considered. The scheme RC RSA-II is not based on the

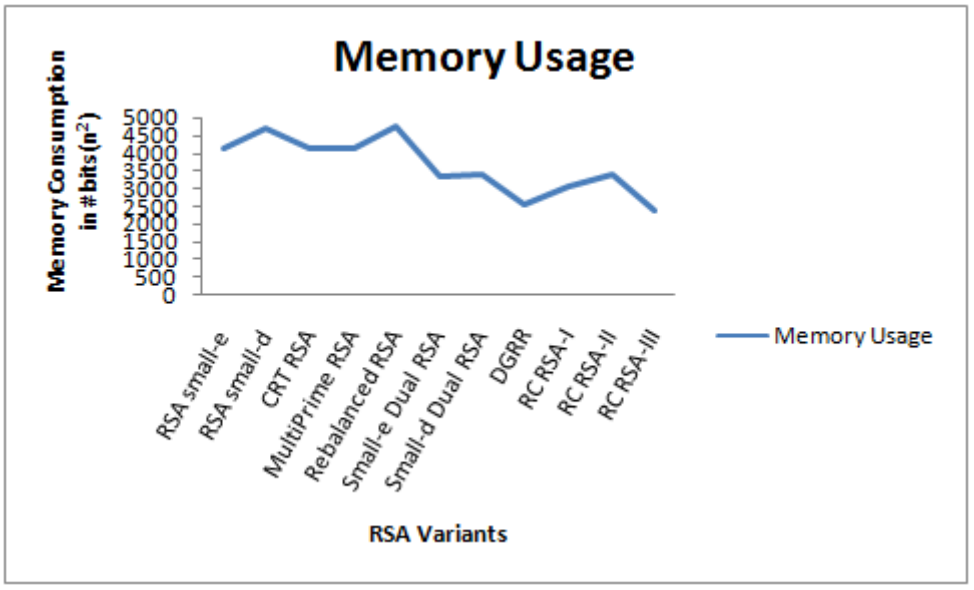


FIGURE 4.2: Comparison of Memory Consumption by RC RSA Schemes with existing RSA variants

CRT computation, so the performance in this case will not be affected by parallel processing environment.

TABLE 4.8: Comparison of the proposed schemes with existing RSA variants (Implementation)

| Sr. No. | RSA Variants | Encryption Time (Average) (ms) | Encryption Time (Std Deviation) (ms) | Decryption Time (Average) (ms) | Decryption Time (Std Deviation) (ms) |
|---|---|---|---|---|---|
| 1. | RSA Small-e | 0.0906 | 0.0115 | 3.841 | 0.208 |
| 2. | RSA Small-d | 3.905 | 0.43 | 1.134 | 0.93 |
| 3. | RSA CRT | 0.0941 | 0.004 | 1.197 | 0.194 |
| 4. | MultiPrime RSA | 0.098 | 0.00904 | 0.62 | 0.035 |
| 5. | Rebalanced RSA | 3.827 | 0.304 | 0.46 | 0.0225 |
| 6. | Dual RSA Small-e | 1.25 | 0.336 | 3.903 | 0.262 |
| 7. | Dual RSA Small-d | 3.9 | 0.435 | 1.464 | 0.322 |
| 8. | DGRR | 0.62 | 0.172 | 0.94 | 0.199 |
| 9. | RC RSA-I | 0.0925 | 0.0037 | 0.627 | 0.047 |
| 10. | RC RSA-II | 0.031 | 0.0009 | 1.416 | 0.0703 |
| 11. | RC RSA-III | 0.681 | 0.0251 | 0.479 | 0.0615 |

Comparison of the implemented results of RC RSA schemes with other RSA variants is shown in table (4.8). From table (4.8) (figure 4.3), encryption speed of RC RSA-I is approximately 14 times and the decryption speed is 6.3 times faster as

compared to Dual RSA Small-e. Encryption cost of RC RSA-II is shown to be negligible as compared to Dual RSA Small-d. In case of RC RSA-III, for the same encryption speed, the decryption speed becomes approximately 2 times faster as compared to DGRR.

TABLE 4.9: Comparison of proposed schemes with existing RSA variants implemented on cPCI Architecture

| Sr. No. | RSA Variants | Encryption Time (Average) (ms) | Encryption Time (Std Deviation) (ms) | Decryption Time (Average) (ms) | Decryption Time (Std Deviation) (ms) |
|---|---|---|---|---|---|
| 1. | RSA Small-e | 0.199219 | 0.0456 | 5.082333 | 0.2378 |
| 2. | RSA Small-d | 5.234 | 1.034 | 1.378 | 0.0486 |
| 3. | RSA CRT | 0.194603 | 0.07564 | 1.724625 | 0.4590 |
| 4. | MultiPrime RSA | 0.217 | 0.01234 | 0.97861 | 0.06382 |
| 5. | Rebalanced RSA | 5.204617 | 0.98245 | 0.661424 | 0.07832 |
| 6. | Dual Small-e | 1.473027 | 0.56748 | 4.905 | 0.8967 |
| 7. | Dual Small-d | 5.124 | 1.0956 | 1.821 | 0.07435 |
| 8. | DGRR | 0.814204 | 0.067013 | 1.333484 | 0.0342 |
| 9. | RC RSA-I | 0.193456 | 0.02367 | 0.987601 | 0.09234 |
| 10. | RC RSA-II | 0.103659 | 0.001205 | 1.70956 | 0.03425 |
| 11. | RC RSA-III | 0.814696 | 0.016703 | 0.578684 | 0.056327 |

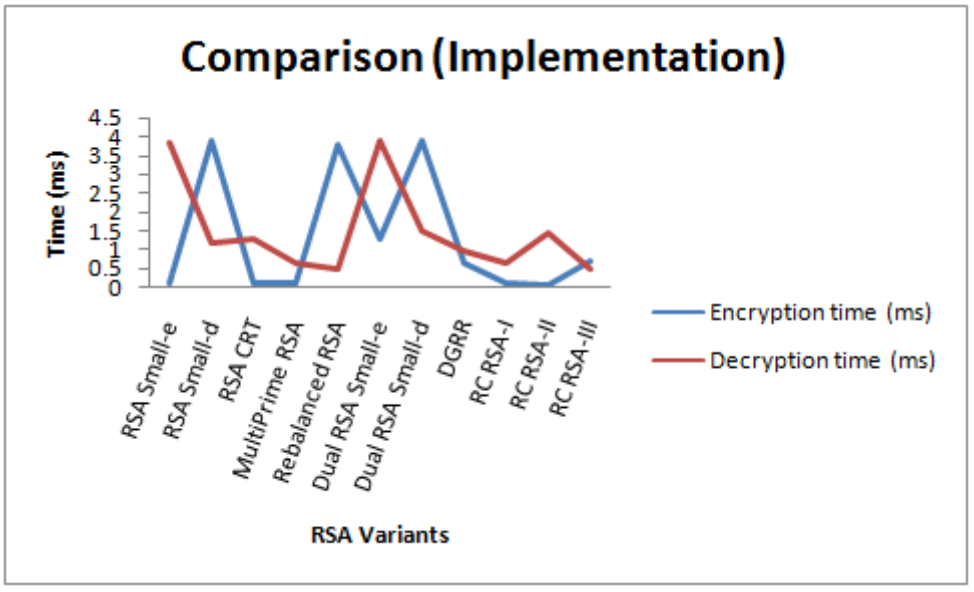


FIGURE 4.3: Comparison of RC RSA Schemes with existing RSA variants (Implementation)

The gain in performance of the proposed schemes in implementation is slightly less than the theoretical gain. In theoretical calculations small computations of multiplications, CRT etc. are ignored.

The results shown in Table (4.8) are the implemented outcome of the variants on a system with 2.3 GHz CPU and 8 GB RAM. Implementation of the variants is also accomplished using other computational environment so as to exhibit the performance of the proposed schemes in a better way. Results in table (4.9) are the implementation outcome of the variants on cPCI(compact Peripheral Component Interconnect) architecture with 1.6 GHz CPU and 4 GB RAM. Performance of the variants (in terms of encryption and decryption speed) in table (4.9) is inferior than the performance shown in table (4.8) due to the constrained environment (less computational speed) of the architecture. But the performance gain in table (4.9) is approximately same as in table (4.8).

## 4.5 Conclusion

For the purpose of better performance in terms of computations and memory usage, three schemes are proposed. RC RSA-I is the improvement in decryption side of Dual RSA Small-e, which increases the decryption speed by a factor of 6.3. The encryption speed of RC RSA-I is 14 times faster than Dual RSA Small-e. Also, the memory consumption gets reduced approximately by a factor of n/4 bits. In RC RSA-II, encryption side of Dual RSA Small-d is improved; the online encryption time becomes almost negligible. In RC RSA-III, for same encryption speed, the decryption speed becomes 2 times faster than DGRR. In this scheme encryption and decryption parameters are chosen as per the requirement. Also, the memory consumption gets reduced by approximately 170 bits.

The schemes of RC RSA are used mainly in the environment where resources are not in plenty. This can be efficiently used in constrained environments like smart phones. Moreover, the dual feature of RC RSA makes it suitable for use in blind signatures and authenticity/secrecy etc. where more than one instance of the RSA cryptosystem is required.

# Chapter 5

# Conclusions and Future Work

The large size of RSA key creates problems in resource constrained devices. In the present work RSA cryptosystem is described, different RSA variants are studied and analyzed in terms of time and memory consumption. Depending on the analysis, few schemes are proposed which are proved to be better in terms of computational speed and memory consumption. Existing RSA variants and the proposed schemes are implemented using NTL with GMP.

## 5.1 Conclusions

In this work, two existing variants, Rebalanced RSA CRT Scheme (RRCS) and Dual RSA are improved to design new schemes. RRCS is improved to achieve better computational speed whereas schemes of Dual RSA are improved to achieve better computational speed and memory usage.

### 5.1.1 Improvement in Rebalanced RSA CRT Scheme

Rebalanced RSA CRT Scheme (RRCS) is based on the communication between constrained computational resources (like handheld device) at both the ends, where balancing of both the encryption and decryption speeds are required. Here, RRCS is extended as ERRCS to improve the computational speed. In ERRCS, the decryption speed of RRCS is increased by using multiple prime modulus. Like

RRCS, ERRCS gives the option to select the size of the public and private parameters. Security analysis is done to fix the parameters used. While maintaining the same encryption speed, decryption speed of the proposed scheme improves by a factor of 1.8. One can get a higher or lower performance gain by selecting the public and private exponents according to the requirement. This scheme is useful in any RSA embedded protocol that is used in the current environment where the need to balance the encryption and decryption speed is required but key generation is not done very frequently.

### 5.1.2 Improvement in Dual RSA

For the purpose of better performance in terms of computations as well as memory consumption, three schemes are proposed.

The first scheme, RC RSA-I, is the improvement in the performance of Dual RSA Small-e. After security analysis encryption speed, decryption speed as well as memory consumption improves in RC RSA-I. As compared to Dual RSA Small-e, RC RSA-I increases the decryption speed by a factor of approximately 6.3. The encryption speed becomes 14 times faster. Also, one gets the advantage of memory reduction by a factor of n/4 bits.

Second scheme, RC RSA-II, results in better encryption speed as compared to Dual RSA Small-d. In RC RSA-II, online encryption time becomes almost negligible. This scheme preserves the same decryption speed and memory consumption as in Dual RSA Small-d.

In the third scheme, RC RSA-III, the decryption cost as well as memory consumption of the existing variant, DGRR, reduces. After fixing the parameters by security analysis, the decryption speed in RC RSA-III increases by a factor of 2 and memory consumption decreases by approximately 170 bits.

These schemes can be used where two instances of RSA are required, like blind signatures or authenticity/secrecy.

## 5.2 Future Directions

RSA cryptosystem being very popular in cryptographic field is always the centre for research. Due to the use of large size of parameters, it is worked upon improving the computational performance as well as memory consumption. In this work also, the focus is centered to improve both these dimensions of RSA variants. Although numerous improvements in RSA have been suggested by many researchers since its inception. Additional improvements are always desired to make the RSA cryptosystem more attractive due to its large spread use in securing commercially and scientifically sensitive information. Present work can also be further taken ahead by introducing more improvements. Following are some of the points for the enhancement of the work presented in this thesis:-

- The schemes proposed in this work are designed with modulus size 1024 bits and 2048 bits. On the basis of this modulus size, key parameters and security analysis gets affected; like number of primes used in ERRCS, RC RSA-I and RC RSA-III are three. This number will increase with the increase in size of modulus. These schemes can be analysed further by considering larger size of the modulus (4096, 8192 and so on).

- As MultiPrime RSA is an active variant of RSA, it is used to propose the scheme RC RSA-I in chapter 4. By using multiple primes, performance of RC RSA-I comes out to be better in terms of computational speed. In literature, the decryption speed of MultiPower RSA is found to be better than Multi-Prime RSA. For further improvement in the computational performance of RC RSA-I, the scheme can be designed with MultiPower RSA.

- Proposed schemes are implemented on the computational devices having lots of resources in terms of computation speed and memory. In this work, schemes are implemented on two computational environments. At first, the environment taken is 2.3 GHz CPU with 8 GB RAM and the proposed schemes have shown to give the advantage in performance over the existing RSA variants. Another environment taken is 1.6 GHz CPU with 4 GB RAM. Hence it is shown that by using constrained environment, the performance of the schemes in terms of computational speed deteriorates as compared to the performance in former computational environment. This performance may further be analysed by using even more constrained environment like smart phones where even inferior CPU speed and less memory is used.

The schemes proposed in this work are implemented using single processing element. Also the complexity is measured theoretically by assuming the parallel processing element. However these can further be implemented using multiple processing elements to demonstrate elevated performance.

- In this work, the processing environment is considered to be free from side channel attack (timing attack , fault attack, power analysis). By opening the system to side channel attack, some of the MSBs or LSBs of the private key can be leaked and this leaked information may break the security of RSA schemes. Schemes proposed in this work may further be analyzed for their effectiveness as counter measure against such attacks.

- The schemes proposed in chapter 4 are primarily based on improvement in computational performance of existing RSA variant (Dual RSA). With the use of small key parameters, memory consumption also gets improved by a few bits. Twin RSA is another important RSA variant in terms of memory consumption, where single modulus is stored for two RSA instances. As the modulus size in RSA is very large for security consideration, this variant can be used for the proposed schemes. It can be further analyzed due to high complexity of Twin RSA.

# Appendix A

# NTL (A Library for Number Theory)

NTL (designed by Victor Shoup)[101] is a C++ library based on number theory. It is free software. It is used for performing high speed operations with arbitrary length integer, vectors, matrices and polynomials. Its utility lies mainly in :

- Arbitrary length integer and floating point arithmetic
- Polynomial arithmetic
- Lattice basis reduction
- Basic linear algebra

NTL, being efficient in handling big data, is popular amongst researchers. For enhanced performance NTL is used in conjunction with GMP (GNU Multi-Precision library). As compared to traditional NTL, it provides two to three speedup gain for basic operations. GMP has the assembly codes for the functions suitable for variety of architecture. Once installed with NTL, users need not to be concerned about the programming of GMP, as NTL hides all its implementation details. With Windows operating systems, GMP can be installed using Cygwin tools.

Few aspects can be considered here to understand the programming environment with NTL.

NTL components can be used in the program by using 'namespace NTL '. All the computations involved with big numbers are handled by C++ compiler and NTL library routines.

For handling big numbers, ZZ is used instead of int (for small number). To use ZZ, header file <NTL/ZZ.h> is included in the same way as the standard header files in C++. Same functions are used for reading and writing as with traditional C++. Following are some of the important routines defined in NTL.

TABLE A.1: NTL Routines based on Basic Arithmetic

| Functions | Details |
|---|---|
| Compare(x,y) | It computes x-y and return -1, 0 or 1. |
| power2(z,x) | It computes $x^2$ and saves the result in z. |
| power(z,x,e) | It computes $x^e$ saves the result in z. |
| SqrRoot(z,x) | It computes the square root of x and saves the result (floor value) in z. |
| GCD(a,b,c) | It computes GCD of b and c; saves the result in a. |
| XGCD(x,a,b,c,d) | It computes the values of x, a, b using extended euclidean method: x=GCD(c,d)=c*a+d*b. |
| Swap(x,y) | It swaps the values of x and y. |
| NumBits(x) | It returns the size of x in bits. |
| NumBytes(x) | It returns the size of x in Bytes. |
| IsZero(x) | It returns 1 if x is 0. |
| IsOne(x) | It returns 1 if x is 1. |
| GetTime() | It returns the CPU seconds consumed by the process. |

TABLE A.2: NTL Routines based on Modular Arithmetic

| Functions | Details |
|---|---|
| AddMod(z,x,y,n) | It adds the values of x and y modulo n and saves the result in z. |
| SubMod(z,x,y,n) | It subtracts y from x modulo n and saves the result in z. |
| NegateMod(z,x,n) | It calculates x modulo n and saves the result z. |
| MulMod(z,x,y,n) | It multiplies x and y modulo n and saves the result in z. |
| InvMod(z,x,n) | It computes the inverse of x modulo n and saves the result in z. |
| SqrMod(z,x,n) | It computes $x^2$ modulo n and saves the result in z. |
| PowerMod(z,x,y,n) | It computes $x^y$ modulo n and saves the result in z. |
| SqrRootMod(z,x,n) | It computes the square root of x modulo n and saves the result in z. It assumes the x is a square mod n. |

TABLE A.3: NTL Routines based on Random Numbers

| Functions | Details |
|---|---|
| SetSeed(s) | It initializes the pseudo-random generator with a seed s. |
| RandomBnd(x,n) | It assigns a pseudo-random number to x in the range, 0...n-1. |
| RandomBits(x,l) | It assigns a pseudo-random number to x in the range, $0...2^{(l-1)}$ |
| RandomLen(x,l) | It assigns a pseudo-random number to x with length l bits. |

TABLE A.4: NTL Routines based on Prime Numbers

| Functions | Details |
|---|---|
| GenPrime(x,l,e) | It generates the prime number x of length l bits with error $2^{-e}$. |
| ProbPrime(x,t) | It tests x for primality upto t trials. |
| RandomPrime(x,l,t) | It generates random prime number x of length l after testing it for t trials. |
| NextPrime(x,y,t) | It generates the smallest random prime number x greater than y. |
| MillerWitness(x,w) | It tests the primality of x with witness w. It returns 1 if x is composite and 0 otherwise. |

# Appendix B

# Examples based on Proposed Schemes

Proposed schemes are implemented in C++ on windows operating system using NTL 5.5.2 with GMP 5.0.2. The examples are given for the results shown for different schemes in chapter 3 and chapter 4.

## B.1 ERRCS(Chapter 3)

In chapter 3, ERRCS is implemented for various combinations of the parameters. Mainly the security parameters $k_{p_i}$ has direct influence on the values considered for the implementation. Two options are discussed in the chapter; unknown parameter $k_{p_i}$ and known parameters $k_{p_i}$. The examples are taken with modulus size 1024 bits and 2048 bits for both known and unknown parameters.

**For unknown $k_{p_i}$ (109 bits) with moduli 1024 bits, the parameters are:**

Size($p_1$)=340bits

Size($p_2$)=340bits

Size($p_3$)=341 bits

Size(e)=170 bits

Size($d_{p_1}$)=280 bits

Size($d_{p_2}$)=280 bits

Size($d_{p_3}$)=280 bits

Size($k_{p_1}$)=109 bits

Size($k_{p_2}$)=109 bits

Size($k_{p_3}$)=109 bits

N=546807751500413547549748214288727006695854094122061567190714864129283
673432718299422846257400275533766456075725257784911415232004425836589131
628300440254617170614215694722221950368465769832403693254687466407040198
493146318675627269996480648456015911999699756490717210454168081532611887
7030738980969969257283

$p_1$=140131610335549274490469539414137690243758874653099987339219022700912
822975032309803573390609589324858 9

$p_2$=177924479329230182115379153651036560965386181542352118932638251354869
583797812939461845405563415893426 7

$p_3$=219312227575676730034034188838467482293132677791628525409123426951827
800756122855934297388504942901254 1

$d_{p_1}$=978165572752705858034373399151870570169770525975168474723383432520777
504232332207749

$d_{p_2}$=153467079805814146726671945177591293168558942352518286316766505225314
92238540601015807

$d_{p_3}$=151431346542554977363582574246358748126635876316811339803814885434883
88843624274159557

$k_{p_1}$=64542947266267434114772274894 3915

$k_{p_2}$=516680659538078143918962991176191

$k_{p_3}$=522330670689239101561043941524347

e=748288838313422294120286634350736906063837462003713

**For known $k_{p_i}$ with modulus 1024 bits, the parameters are:**

Size(N)=1020 bits

Size($p_1$)=340 bits

Size($p_2$)=340 bits

Size($p_3$)=341 bits

Size(e)=17 bits

Size($d_{p_1}$)=339 bits

Size($d_{p_2}$)=340 bits

Size($d_{p_3}$)=340 bits

Size($k_{p_1}$)=16 bits

Size($k_{p_2}$)=16 bits

Size($k_{p_3}$)=16 bits

N=623930583191505716544276929030454097485074381991891360264262309582718
671081059599021474477208680437464105540887851902256232576212711160865335
952591416861937707947915880458077322391261995295211684269069870544037526
014404307311948796228419668103813706124174667162572168463445068649334366
5961154894123683878639

$p_1$=130666666581940720138003408590632851063799388698881653713472877462483
638743898607089395085520737267892 1

$p_2$=203425490200859248329791433960116469116535495202519912714639253759171
650477204144731224646034039692472 3

$p_3$=234728644447538656665127735219891729385394273287031739034014365805439
854248985822287471836038930866353 3

$d_{p_1}$=742266098567079828191361963230033479142192569368549839952833018760517
476661519636744295533645349611113

$d_{p_2}$=165150321675189849499288692422325661165668058619486312707250806956535
801688971477555724835083801758970 9

$d_{p_3}$=136090823551171496060314914554377314211160507836601385294954664655222
212519625773066467298685799718156 5

$k_{p_1}$=37229

$k_{p_2}$=53206

$k_{p_3}$=37997

public key is e=65537

**For unknown $k_{p_i}$ with moduli 2048 bits, the parameters are:**

Size(N)=2048 bits

Size($p_1$)=682 bits

Size($p_2$)=683 bits

Size($p_3$)=683 bits

Size(e)=551 bits

Size($d_{p_1}$)=262 bits

Size($d_{p_2}$)=263 bits

Size($d_{p_3}$)=263 bits

Size($k_{p_1}$)=130 bits

Size($k_{p_2}$)=130 bits

Size($k_{p_3}$)=130 bits

N=230945687488072239569995883101517824042247547406142682237757510990887
920818416998498899251002203638208046465940441284945807250505148504487094
599802392648662509628125353352412158982670032099687777854432021862037776
752739816529077194469076315102059154459689995904926047214347158150839941
611848443662111872417058378455433546586876381037308817586355949893146154
862009668337251924783064653774673244255188895515905984574200330752291012
519410220387271587452208283228585440144824536812694480444062308341707370
081736850040439548178455212652360915819092355385091122376420673636822021
063284086162186851706797213142555893977620 09

$p_1$=168106004333255329822193035986640680280283040895660099537754399006379
866416619871132330401395053501914524704663632869819760915673039813710351 6
964348329397337237116681930929396844188523879674774970857517856 9

$p_2$=349541670694910670196984350695795071669456561796761488543829900772556
696507968700899585167314575380752172983283015392078102501422454222015247 1
083004249358357711526701909676660810181427223468411996554594094 3

$p_3$=393031784870080743129446499667606325758276446018048720450002764089215
141891262014667035694356225579875469692433936299739126725241710328273067 4
135091608398446133232915105007954502924514482789500140121082012 7

$d_{p_1}$=544660045271441586508832141968278685581533050975810753757715811965874
4078866433

$d_{p_2}$=821442126853818675906990269049422528298182034120489259816274356927675
9724998625

$d_{p_3}$=960072963719369427850710789074584245335131804782505640488373701058521
8528543113

$k_{p_1}$=119409782518804700708366378175665300 5568

$k_{p_2}$=900272908708990372546003735607040656124

$k_{p_3}$=866115137340908407733362912017004323472

e=368551018048978647679839314549635633878605587931293010583613896508361734 6
086082863365358130056307390177215209990980317284932211552660930305235775636 1
64742230126362625

**For known $k_{p_i}$ with moduli 2048 bits, the parameters are:**

Size(N)=2045 bits

Size($p_1$)=681 bits

Size($p_2$)=682 bits

Size($p_3$)=683 bits

Size(e)=17 bits

Size($d_{p_1}$)=681 bits

Size($d_{p_2}$)=682 bits

Size($d_{p_3}$)=683 bits

Size($k_{p_1}$)=16 bits

Size($k_{p_2}$)=16 bits

Size($k_{p_3}$)=16 bits

N=343356313114907165353554369555770684728526534954679941901247595187146
601285863083434192850996917345964023373436778664002729308535880961860127
605641250081568662591963531758558057545085445471962437878366935793583469
478587102002908638942442860843457139858745134310699987501307419559925670
486073533296851797764391761852360865858764807340396961750731205171611050
031606526776758433068704361465421206499775263280312616571533231204896202
805137035980533803683375836383281582723863783943821606588141890590818797
855523470390751446102946107891626291167384264562171909820291083737681625
877708130259315266006464885990422470590093 9

$p_1$=710506655238816341547681058057003745880102150238814006294297601471036
902115510803685741516085592301149941338929280469014179932197634683175528 2
676102068988473679284067596011540402821685650831129717111777673

$p_2$=187299273339606109606412657344687220776071452911101060375223270077764
821205499087236114454500865085594848251734935256122766877235809287529631 5
913268820673837623759064865064958481810069540296657838460263237 3

$p_3$=258012528142335570018758082562891997908332397642971456958808729030693
982597029831013098120092206877632294079083642041607526213648892759045387 5
121025799503089375703816299540426385182602911328097013362535079 1

$d_{p_1}$=539593470017644490158401843563379319162373075082567873434545069509078
058075487186185616167029496315254510891880741375158670118945644040023382 1
037809972652002868079506422153690112901733955066099886782846305

$d_{p_2}$=132072894988223707878620445456307590120460046285010197340741763890683
822609666742732236100002265562973506908424654210464339620332002603179987 8
653278178888262479313634505840165499181969630372605531055192410 1

$d_{p_3}$=20263217455004061505509068937412532054170725707132979675099387038176631954879114701991486093574451060826977509551941470527143776047897078087332724903169266827924298552716380936882272391450029319815946668373

$k_{p_1}$=46213

$k_{p_2}$=49772

$k_{p_3}$=51470

public key is e=65537

## B.2 RC RSA-I (Chapter 4)

The scheme RC RSA-I deals with small value of public exponent; two examples are considered with small size of public exponent ($e$) for modulus size 1024 bits and 2048 bits.

**For small $e$ with modulus 1024 bits**

The parameters are:

Size($N_1$)=1024 bits

Size($N_2$)=1024 bits

Size(d)=1024 bits

Size(e)=17 bits

e=65537

d=161461816393387084600238062954452367332075089168849462538531109613337802394048843499736224632930914069189618840110283677084021149990200033600751565713434101018980185825043303167928186477196509431667458559007168130246691745581725033449012793448319080726272518204281507746065378832318595370735002186243435855873

$N_1$=12465511098121535863073463777973264534259500893953076084237717145803081188402928453543753293017669554631912463638708691108808309554032888369244490633327509394703671635354329158859958718966750933625730802132332

00787501457702506610051349730798675801567995661016408955728637877220352
93590367074784510552927295 4801

$N_2$=14321301241031573954425349084893277389890381548910360581403899592260
33937242688806902638175856484586855592927967749338923766727242936075 10
92148928504984262037867274907404775032201735386946431919385408489667904
51570354024478251610577658337622061605376827841296638647702530565932544
00067532523290981217324610769

$p_1$=28762528590785929135827299027153588249878134946676933278338516089238
00566116269671764892004748707943553

$q_1$=62352534203750375569814710284614969113518997292754256382658737103323
89712465947208820023051461524969217

$r_1$=69507059661108855234412676749222932871427815107335299575632135038566
28428304775383429045967913519946801

$p_2$=54937758663987043790864276572514304707539705482602224491399075224895
85419198657101941572198549747497617

$q_2$=43467667179535513087687033328389972368102967473550956550351776480466
01158710577314691897206830063390657

$r_2$=59971541172470897838140312764791227994626202798053151736437422668872
56905706787596962946658967686825601

**For small $e$ with modulus 2048 bits**

The parameters are:

Size($N_1$)=2047 bits

Size($N_2$)=2047 bits

Size(d)=2048 bits

Size(e)=17 bits

e=65537

d=16556992671056676776053525938995833366531480772549453351428016417992
19201218347392080812840055824202689184651124534203052860599900674405005

616037789856776015135976153321344337857648197839489073734615141118800409364715642281984998850515210125052631161799079394932675050605416748665386539355691446853686368513279658262197375114410689868215054518454090291563403894933323373864295144572746380471208245963051223579131438065830408239926605595623635048420380744232874873778163395138216422888519792930937590337117127509320684364481936628035532868084743517516873653396771218257277727963730928627563335201082522509837422604083325777279637309286275633352010825225098374226040833

$N_1$=143345349769219982809614511541119967679776698907576623463306548652005903445595435856694009219184853987742205184745433255139270701591189305996272383198821012053114308383604512497261076834006969041910582186472964753868710430184500564978298897917837011937601009812387874871148456842297190224756178687824744633925566217400699947254170307233246395020104436144471394008011524376041085430363690226659455767386926076638069240253286362287263268118094534130388229175028817288160615825734409012946596712266978732092849694645096682850249950156706392182490501309114298936651350796472920283784431086332878894480095008039351882797493

$N_2$=138869132647756715794136006739866637403359908801168896605049772452066125048307907848678275741301992336175861272042537645544197619200003174963600640704706454866987213852697719253342406948996129188545230878083471038018815832421184505233820570337014022393535016845303186289112654875446858526956924991320701526326690265267554480233275627898134511308334404323863504175951640330088059715529888790744169653670779610897892661125605689162080733657382807563347829526592068253719855245937436797780354209555160748920866861998826482423282389319216274568742619456348589275851154941617465516050024197461754947611574168300963738314373

$p_1$=217230494248112584465642472927815694993090455308633347106847860511853874728342799006632529015333602715134896834997154432258143066302459704086217357894235673260101583619397816576972050741567990836372590530173

$q_1$=233341128014410505690730435683543079924292714285892265999761548715709397268168388648186641329450254035710094864041132860033705276475246414888134449516859091760890657549248564151442558028741893588762432921413

$r_1$=282794868969541668178710610789985205355213612963517607816730112358336959842182736559857444480758472888551999839289975735386798133926280616038664458550848994562068697913848823868294462149265928847889757300173

$p_2$=171826851771774221869850702546477986576675712426442856394579171916591470207648112647040140619046336796388867108486407082227586354001276319716356364516111203798866719149818263090098044297162433713517175891410

$q_2$=243583755972809682816643149900196899268557379262176219130462873411864015533463215800112492900750837866349676398700676400099643679481120395887106639603423448988583932500023724264013323613864217935292102550090

$r_2$=331792349004776734823473426169636949175118040892576664923774152917245290810217815790595687218306126975869479608763602765265048278570392841432768906233144573202540167378748519506454047554367573195599109132730

## B.3 RC RSA-II (Chapter 4)

RC RSA-II is designed with large value of public exponent ($e$) and small value of decryption exponent ($d$). Examples are shown for modulus size 1024 bits and 2048 bits.

**For small d with modulus 1024 bits**

The parameters are:

Size($N_1$)=1024 bits

Size($N_2$)=1024 bits

Size(d)=342 bits

Size(e)=1024 bits

e=137115843000522618943082103867035291367040024944224685657635727094645673053524307397672064224348940761042041072952371533477124151463401988143625787932037515249038486503637862897081464023747537434380067802980210477098637400308347055163906757495221932255994492388515473506608238257989803989153533874719652581103

d=851772374707640182339854707676496133107034523204571853857931115259723617970360219466831595595090989876 7

$N_1$=112945241198014717631593253869511037085938345110942955694282264911517922512996046119006042025991750387750984110094729892375171113649257758

810699223561614246451332975044944991964121356389629809468925630968416895040524478216670516071207378888081384444392958512024806137942351530999784010097941740621204032484641

$N_2$=138105392762719807666930156060444482714266321371604142693122908508352023603621880786504068009324932635925217137710290405086432450461912342887297454350769314700554613185314074288708076744506148552539552150224727959199795734922158107538402129350556769458716559193462067213119255308946534371148993341509409243681

$p_1$=946823457254939832395151004277979541433434523374889248347478276911195513302542545625266525825447564205883575952865541560483466250675212236330780164738208 1

$q_1$=119288596340303071804803802269686634036861824228175565618371697367024400585385633284625838414093044538049498319082126566107777982288052927067671995174417 61

$p_2$=118834083987031564988804839759351422466194284638560209358833502323592819753002396364598334568317800888363965665984460862678748230583003114831245850024876 81

$q_2$=116216987693355159103941590686314944378268774033059807450741920205172057150351634953981722396884443328684626345265140285629434193497678448006359283030760 01

**For small d with modulus 2048 bits**

The parameters are:

Size($N_1$)=2048 bits

Size($N_2$)=2046 bits

Size(d)=684 bits

Size(e)=2047 bits

e=131477077521912567876205093018134085534719635769915454007968145250238442241259197371224520349065298229275245766769272607215247549903033457188685655512712905593859296975672079795843085637708037754704532688869553132840862923419412744620516395005746189927395851802968323440148004788178517 3

736722595327073946252598464749283760239948020562964816176320229104744985
319106841425173131472623515295908881461317085682378322879082470766906828
451616430065682103026754316653727390401114928257241038710065797271205672
043937787680033049539266769995069341102141122482527870954093688581334879
986232301831721978675427920931604119257241

d=439221564036321321005446608478195346099376142711410945992027487755751 8
531116082047382685771849187631587885661871967099551637497620844530871760 8
740605711228873399435298756497970294155897510695337243245876073

$N_1$=169300132433222490624288318997289114230232480590984895607020293863862
112627876356364757734788433542030648540404058022295576360914086953863296 5
963085684707402988895751239419458454799525985501047621617129001688008052 8
689423043423262602714040506151442112339579007751710938604203423357110188 6
933615496400623414330394607974526250417285581022563828535334394621333387 6
155068471319096448282706310765389370935661060800339370356544186484924250 5
278227351713165762521497033182034894944976849331545693641823996938945839 2
035943971244073880896018989442462259719867888680167083678006540181884033 7
176393710204706493288235309171966299 7

$N_2$=755636804019187817263859409870139736891664178454617706126432944694600
405174606544921351266891519063617827598463978607863588025692021421772058
621153482605347358786834894254642655493328290660712876265900234869359704
609212445768933053286549116200111002579485281812175260319183295212736462
561070709763854397314889783895368429006124830866006153494000333774068794
279953306231081779619847191093151403548697225852173152364326934032223527
234898254286915585107297439436894305504694781830902907237000153198345857
507401261262871550127506206405731812047822679165361826204636928836795277
853933606018036575797740592352887994159326 9

$p_1$=132129669333399197946206061479023624547501407158398949165458022466245
338983075184311538600552851419204771009962366484188037651391546877054846 9
302044410568710943738546249806241466803909018466931913426057320072371603 8
464882804542183753054491699320416199661219221642746362705420675976122141 2
427672556854877880729

$q_1$=128131806646720711114577847110737083067538168879833921036028601951892
829691597363257485177868972792338348475462132384862816461671775938519124 4
270317791985386024965594884551139948726217724573116849829883965696080334 3

6967724522637338964507160709425265208209066260211871547358146356713629745
267932809940079468893

$p_2$=1176397756735480890341519955869312394785025226060968870783334991187588
1067984005539782608229586476969181206898777939591939787161265896689397599
1015336313065104897297777485476308182237729017876079702964803989798107110
9762621036200726374532700267032762919512406337524487221694412769580440590
12199029361368079373

$q_2$=6423310480598754271533824536055490474624272092758739748094902342072201
3968078303117507205849584298749158157440720469463838827479984367305936769
1334904251781124504015704467484402500256142295452019902031007718233911781
3778871251703356029338034405053063641506695616460257283658624702083057391
7450360189646900553

## B.4 RC RSA-III (Chapter 4)

RC RSA-III is based on choosing the size of encryption and decryption exponent. Examples are shown for the modulus 1024 and 2048 bit-size. First example is for small encryption exponent and large decryption exponent and the second example is for balanced encryption and decryption exponents.

**For small e and large d with modulus 1024 bits**

The parameters are:

Size($N_1$)=1024 bits

Size($N_2$)=1022 bits

Size($p_1$)=341 bits

Size($p_2$)=340 bits

Size($q_1$)=342 bits

Size($q_2$)=341 bits

Size($r_1$)=342 bits

Size($r_2$)=342 bits

Size(e)=17 bits

Size($d_p$)=340 bits

Size($d_q$)=341 bits

Size($d_r$)=342 bits

Size($k_{p_1}$)=17 bits

Size($k_{q_1}$)=17 bits

Size($k_{r_1}$)=17 bits

Size($k_{p_2}$)=17 bits

Size($k_{q_2}$)=17 bits

Size($k_{r_2}$)=17 bits

$N_1$=1076762907520463658467291654342037015297146473553663476744016352319
9757485072055152543974259411175587435180118586934268868473309315317734 9
7663305372757424165449857955767048064303671367397536322195952180391816 7
3724741561922795411868862575624169262298624564467175665366742758065449 8
501472349913610737324953600 01

$N_2$=4231173759872873223360347256938584748122337616567603381871910391340
1636809597404017595839637722911798162900594298787967017605473522924623 0
9502028865077920532599868967996584013485580914232902311577441075695400 3
1515354844756997146962642975134260563785220672234419643280163884226944 7
3830788097677222136649728001

$p_1$=3243804365390713566106440759154583949717057209075476829033150205484 7
4904620357851825943860051297280000 1

$p_2$=1643820620744352042743316390508506328185818882819291932293273271193 22
9480132648786694136266861747200000 1

$q_1$=4795681926004760520618758598598860119667430874596775728701714128809 5
5773123340692911970448303456256000 1

$q_2$=4217267618487153427860032967571814418097883883155354485161785331861 3
2947920607184167285121758317772800 1

$r_1$=692173649152522336875134463238362621588529987792401102315752661118056321591860606398145310621696000001

$r_2$=610344837227320218586039257637740940850952779888449253398774611798455233851031561588619231232000000001

e=110927

$d_p$=193104132689584835376273419033033392338714535506503477426779840633196068825491816648121762947590901086 3

$d_q$=437705897496935799354102784314497817443552400648783399782448227444350665699960380368887070488114933086 3

$d_r$=514011956953122571601947239258792908429464041616549990563705188633965486230850175809741721695008501086 3

$k_{p_1}$=66035

$k_{q_1}$=101244

$k_{r_1}$=82375

$k_{p_2}$=130309

$k_{q_2}$=115130

$k_{r_2}$=93419

**For balanced encryption and decryption speed with modulus 1024 bits**

The parameters are:

Size($N_1$)=1023 bits

Size($N_2$)=1024 bits

Size($p_1$)=341 bits

Size($p_2$)=342 bits

Size($q_1$)=341 bits

Size($q_2$)=342 bits

Size($r_1$)=341 bits

Size($r_2$)=342 bits

Size(e)=140 bits

Size($d_p$)=219 bits

Size($d_q$)=218 bits

Size($d_r$)=219 bits

Size($k_{p_1}$)=17 bits

Size($k_{q_1}$)=17 bits

Size($k_{r_1}$)=17 bits

Size($k_{p_2}$)=17 bits

Size($k_{q_2}$)=17 bits

Size($k_{r_2}$)=17 bits

$N_1$=695508146883890427022009175798857986886957417292485689597467064430064511644880499222383579694154325987093011363146805648093288135472368771282672980712977842571175716304435801848096386836547364046214298526177472261793482887750319782119704534850685261543879822351336148380590072393084196447275225792846887321 61

$N_2$=139734538798983805759053968136503561508307936826214134692152213146460118079855197332158221756251614881526421423062454993828328671308279833249730607598517100759112281910188706660788736448595134863088425002509241912863837214881806630632260433709140387972535764211673422765143197099282468162353483834522573701121

$p_1$=400073368767492494817638925146173553995267383791122922651372162400644624452012074293473421580607488000 1

$p_2$=491759010329226467791347162882429659702615771619520613002863071688020450830361245294896263486914560000 1

$q_1$=413903598711509384455747412710072309748234453447700007649948217196132103913409671948435778205248389120 1

$q_2$=508689040612500109328142321050005284463060179718626156647089587064896118089580584990949872936545157120 1

$r_1$=420013622136251727763711277038252333241702512180562271329070882312514437175005972469799463659592400896 1

$r_2$=558597591255565484052428092078201646850588423587074194300964796627283286556060297851783218669537902592 1

e=985672759416490995281855320902478749181913

$d_p$=519496861198279173859998012144502749199603733310073187379097167977

$d_q$=347618085020630966257576800517246910503863616447092005135055644777

$d_r$=517197344515682344510707705491973976471768460137594513521972734057

$k_{p_1}$=82782

$k_{q_1}$=127990

$k_{r_1}$=121374

$k_{p_2}$=67357

$k_{q_2}$=104127

$k_{r_2}$=91262

**For small e and large d with modulus 2048 bits**

The parameters are:

Size($N_1$)=2047 bits

Size($N_2$)=2048 bits

Size($p_1$)=682 bits

Size($p_2$)=682 bits

Size($q_1$)=683 bits

Size($q_2$)=683 bits

Size($r_1$)=683 bits

Size($r_2$)=684 bits

Size(e)=17 bits

Size($d_p$)=682 bits

Size($d_q$)=683 bits

Size($d_r$)=683 bits

Size($k_{p_1}$)=17 bits

Size($k_{q_1}$)=17 bits

Size($k_{r_1}$)=17 bits

Size($k_{p_2}$)=17 bits

Size($k_{q_2}$)=17 bits

Size($k_{r_2}$)=17 bits

$N_1$=14977636178906720436955542111531234961123757983058194812037566326967353085409550057189767765672017233669517924793677268588405174654419915989844349695917422043488761321668607444136606134636851345427748501055277403849422382782767424919634787074460135333314046548836659400760334107761189187141252898451810552300768797415439061326164361589402908940179573458096149013682781481304329609025367751879025195640826775193234644117420298516439264960333249510896822872665892850771849170800416639033152393252128950606296241887477426038645269981462329618533836325475321329433537597451836351560727735424980386849208737162330112000000 1

$N_2$=18354781480113140012647268431547421980276961060107367790828694799666811415291533960628407144879720049196948203158815768485437634410729888771260041441737781793350971856619486339349804908599516893734338168877060062905581866524300021739808236344896054294948344405050161454237509541122023895361093225208654886305328798607138814244428296659302881343975512842107678040106152403516341326539644718035034247388131242135817349040248364494000832838915996398929642414614847811237704062080088009164186543875577978878848512761982782109011124061034168474016026160892387274621004808604865926955721416236130835265899675906048000000 1

$p_1$=175349101438144348689433214841958296138729960077538463121349751204724176638630342164244682954343895698709317928774559500526826066838395489713946508091620755123011890195352890381197307927248996873011200000001

$q_1$=242703501225122357828479870819833010121366944698297327599264485505408658160680792446340329036236056913326815104772366031831871686143768681046150824156519220607919151267535104189163512599528079360000000000001

$r_1$=351935997638723787500871088094257658038737137746477833393197789425422793561779958824663604026136484781775204095477643705013475013563822669928037769957047869572529644132901555030706050710896640000000000000001

$p_2$=110958761673185320536542292909421815857578523370675454546183002947604469542544214968287228899481528764258472360370064419770810295406452064581533412236433603203052817934079337711204100692356046310604800000001

$q_2$=348366565570383687087463008515755793611222671987399606463532646460794400331277404811806574448546379149618839975274747455565500300915006883584502931767805381603901556354977232104516720372807106560000000000001

$r_2$=474844213702973398622162956489447295453549435926383573529330864329945756330614915309010320454424742578156154385019268212759415522004482122853029528432911698393118762320945001804886455186326814720000000000001

e=128903

$d_p$=108915239798503257660164307886831438668173378304369910617084080121932351665582468799969083616537228404403778416990014824861957098496411823459786359641464569324833944907264567320085208695710093509340471045 67

$d_q$=234586585452940930828343514620952455699567507147624652958019291855813096185158927972773591578805169801674239913220007514472095739587320911917504041669387390006298246702751167165514071525103391415414071045 67

$d_r$=298382359680851844320110466596163246252899500477390514230668021773935488412060294485231809630562548186082934494826037603729258879020372310583115922849474780027172523122010699100841740456719176375414071045 67

$k_{p_1}$=109288

$k_{q_1}$=124592

$k_{r_1}$=80066

$k_{p_2}$=81000

$k_{q_2}$=86802

$k_{r_2}$=126529

**For balanced encryption and decryption speed with modulus 2048 bits**

The parameters are:

Size($N_1$)=2046

Size($N_2$)=2046

Size($p_1$)=682

Size($p_2$)=682

Size($q_1$)=683

Size($q_2$)=682

Size($r_1$)=683

Size($r_2$)=682

Size(e)=300

Size($d_p$)=399

Size($d_q$)=399

Size($d_r$)=399

Size($k_{p_1}$)=17

Size($k_{q_1}$)=17

Size($k_{r_1}$)=17

Size($k_{p_2}$)=17

Size($k_{q_2}$)=17

Size($k_{r_2}$)=17

$N_1$=69929224680236768492614443713380514064200795299225502152271462652766
03181563336718066816910090390466490386882840827208252354007113278572646 3
08496736700937999252782034734264087025991526605717864260311556894324347 9
88537024382010224859594638700540627454640598938853259367812990475584720 5
06903045279678258760437942787972059625596660101731115148191068427751753 6
44740104109761707620816536604105040705294504484430865698940424829951868 3
62706344620011294796126834349474293298733719772859441739823679213103095 7
77913109755504989518323500958619941771440123635179614091165119470036698 2
63989531947113326240217247534387885093120001

$N_2$=41383614725613940637278099854100899061692155655341470742201648447174
43211368630494575089229209123784256214283891250792168465260307671593898 0
36734289015498225515284515093806203504946613173902953747099495608600553 8
51181976062279692406337845239301232323596079583410285130140903716509718 6
62581579319691564602955702888891138048205779409781273342754656393782186 9
05487656836436619926998106921840858921303544505526420091769480310211369 6
86341423360194580731999117235762967321261205060841061358637873541911821 2
87187617807025328455436024451775922665006660526202652479830003801346728 3
50144202087624817957753456602735905255680001

$p_1$=13330611271271339057569544242234184329423838406550876072033692765016
86355253704258789183126342678368420155359699949215310421335889071964860
92587245869258738529221612621564481250598728905645840645357568000 1

$q_1$=22172109626669093714948866907606003828330834725639933372359425701069
84147947883286877344601034486760011788762780749971103199773184983548360
69361876181980931005361755273585537851223766567056593214789184000 01

$r_1$=23659285860649098669708034291292587297332161921262428266194951313544
13555528948112869062696088372824310127363259832915738549976575769500114
48605272685085270592770501865403590767720261542848148971664179200 01

$p_2$=16122728635987804689206419735651288830712182793057154847283604785262
31571116855549218306563506354887778381308586764187693388614694422092517
24737699164626099734519788087677846705798398925895113955344384000 01

$q_2$=19569694545082774100798065092914419687635242516167481480329533014051
98170420403294367772255055604923247681986840582133302161942142605408363
93506804977344204365258162880251923124744140923702228693390912000 01

$r_2$=131161335738040338850452788522479113082769070384957944802751030823397236055802429398519734412934828782362537109341706259602678575142978280538198874834653170386183917056485726875905963751866945169837260800001

e=183573436181321785960784534401717197066066076250286653834961622305778888984596413857876119 1

$d_p$=826646543336234049133958775032447412280524397923098199781818922361331415949124273510820927692861411150281882593961405911

$d_q$=125556264134760236382780467862738103930429936235064366722234028852380302457818762724986338247615145210618187845483820591 1

$d_r$=846470278641445963657391682517369155300301131046050034374058496135670448452868024215728346106306758678781064568939965911

$k_{p_1}$=113836

$k_{q_1}$=65678

$k_{r_1}$=103954

$k_{p_2}$=94122

$k_{q_2}$=118472

$k_{r_2}$=117778

# Appendix C

# Publications Based on the Thesis Work

**Journals**

- Seema Verma, Deepak Garg, Improvement in Rebalanced CRT RSA, in International Arab Journal of Information Technology (IAJIT), Science Citation Index Expanded, to be published in Vol. 12, No. 6, Nov. 2015.
- Seema Verma, Deepak Garg, An Improved RSA Variant, in International journal of Advancements in Technology (IJoAT) Vol. 5, No. 2, pp. 161-169, 2014.
- Seema Verma, Deepak Garg, Efficient RSA Variant for Resource Constrained Environment ", in International Journal of Computer Science and Information Security (IJCSIS), Vol. 12, No. 8, pp.106-112, 2014.

**Conferences**

- Seema Verma, Deepak Garg, RSA Variant with Efficient Memory Usage, International Conference on Advances in Engineering and Technology (ICAET'2014), Singapore, pp. 209-211, March 29-30, 2014.
- Seema Verma, Deepak Garg, RSA Cryptosystem: A Review, International Conference on Emerging Trends in Computational and Applied Mathematics ICCAM), ITM University, Gurgaon, pp. 189-195, June 2-4, 2014.

# References


[1] Data Encryption Standard. FIPS pub 46. *Appendix A, Federal Information Processing Standards Publication*, 1977.

[2] NIST FIPS Pub. 197: Advanced encryption standard (AES). *Federal Information Processing Standards Publication*, 197:441–0311, 2001.

[3] Whitfield Diffie and Martin E Hellman. New directions in cryptography. *Information Theory, IEEE Transactions on*, 22(6):644–654, 1976.

[4] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[5] Anil Saroliya, Upendra Mishra, and Ajay Rana. A pragmatic analysis of peer to peer networks and protocols for security and confidentiality. *International Journal of Computing and Corporate Research*, 2(6), 2012.

[6] David Mudzingwa and Rajeev Agrawal. A study of methodologies used in intrusion detection and prevention systems (IDPS). In *Southeastcon, 2012 Proceedings of IEEE*, pages 1–6. IEEE, 2012.

[7] Ahsan Habib, Mohamed Hefeeda, and Bharat K Bhargava. Detecting service violations and DoS attacks. In *NDSS*, 2003.

[8] Nwokedi Idika and Bharat Bhargava. Extending attack graph-based security metrics and aggregating their application. *Dependable and Secure Computing, IEEE Transactions on*, 9(1):75–85, 2012.

[9] Michael O Rabin. Digitalized signatures and public-key functions as intractable as factorization. 1979.

[10] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2010.

[11] Paul Zimmerman. Integer factoring records, 2005.

[12] C Vuillaume. Efficiency comparison of several RSA variants. *Studienarbeit (March 2003) http://www.cdc.informatik.tu-darmstadt.de /reports/reports/studien.pdf*, 2003.

[13] Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *Advances in Cryptology–EUROCRYPT'96*, pages 178–189. Springer, 1996.

[14] Ellen Jochemsz and Alexander May. A strategy for finding roots of multivariate polynomials with new applications in attacking RSA variants. In *Advances in Cryptology–ASIACRYPT 2006*, pages 267–282. Springer, 2006.

[15] Nicholas Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Crytography and Coding*, pages 131–142. Springer, 1997.

[16] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology Eurocrypt 94*, pages 92–111. Springer, 1995.

[17] Gustavus J Simmons and Michael J Norris. Preliminary comments on the MIT public-key cryptosystem. *Cryptologia*, 1(4):406–414, 1977.

[18] Hugh C Williams and B Schmid. Some remarks concerning the MIT public-key cryptosystem. *BIT Numerical Mathematics*, 19(4):525–538, 1979.

[19] John Friedlander, Carl Pomerance, and Igor Shparlinski. Period of the power generator and small values of Carmichael's function. *Mathematics of Computation*, 70(236):1591–1605, 2001.

[20] Gustavus J Simmons. A "weak" privacy protocol using the RSA crypto algorithm. *Cryptologia*, 7(2):180–182, 1983.

[21] John M DeLaurentis. A further weakness in the common modulus protocol for the RSA cryptoalgorithm. *Cryptologia*, 8(3):253–259, 1984.

[22] Nicholas Howgrave-Graham and Jean-Pierre Seifert. Extending Wiener's attack in the presence of many decrypting exponents. In *Secure Networking–CQRE [Secure]'99*, pages 153–166. Springer, 1999.

[23] M Jason Hinek and Charles CY Lam. Common modulus attacks on small private exponent RSA and some fast variants (in practice). *Journal of Mathematical Cryptology*, 4(1):58–93, 2010.

[24] Johan Hastad. On using RSA with low exponent in a public key network. In *Advances in Cryptology–CRYPTO'85 Proceedings*, pages 403–408. Springer, 1986.

[25] Johan Hastad. Solving simultaneous modular equations of low degree. *siam Journal on Computing*, 17(2):336–341, 1988.

[26] Daniel Bleichenbacher. On the security of the KMOV public key cryptosystem. In *Advances in Cryptology–CRYPTO'97*, pages 235–248. Springer, 1997.

[27] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[28] Matthew K Franklin and Michael K Reiter. A linear protocol failure for RSA with exponent three. *Rump Session, Crypto'95*, 95, 1995.

[29] Don Coppersmith, Matthew Franklin, Jacques Patarin, and Michael Reiter. Low-exponent RSA with related messages. In *Advances in Cryptology–EUROCRYPT'96*, pages 1–9. Springer, 1996.

[30] Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on RSA given a small fraction of the private key bits. In *Advances in Cryptology–ASIACRYPT98*, pages 25–34. Springer, 1998.

[31] Michael J Wiener. Cryptanalysis of short RSA secret exponents. *Information Theory, IEEE Transactions on*, 36(3):553–558, 1990.

[32] Eric R Verheul and Henk CA van Tilborg. Cryptanalysis of 'less short' RSA secret exponents. *Applicable Algebra in Engineering, Communication and Computing*, 8(5):425–435, 1997.

[33] Andrej Dujella. Continued fractions and RSA with small secret exponent. *arXiv preprint cs/0402052*, 2004.

[34] Andrej Dujella. A variant of Wiener's attack on RSA. *Computing*, 85(1-2): 77–83, 2009.

[35] Alexander May. *New RSA vulnerabilities using lattice reduction methods.* PhD thesis, University of Paderborn, 2003.

[36] Dan Boneh and Glenn Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. In *Advances in Cryptology–EUROCRYPT'99*, pages 1–11. Springer, 1999.

[37] Johannes Blömer and Alexander May. A generalized Wiener attack on RSA. In *Public Key Cryptography– PKC 2004*, pages 1–13. Springer, 2004.

[38] Jean-Sébastien Coron and Alexander May. Deterministic polynomial-time equivalence of computing the RSA secret key and factoring. *Journal of Cryptology*, 20(1):39–50, 2007.

[39] Benne De Weger. Cryptanalysis of RSA with small prime difference. *Applicable Algebra in Engineering, Communication and Computing*, 13(1):17–28, 2002.

[40] Subhamoy Maitra and Santanu Sarkar. Revisiting Wiener's attack–new weak keys in RSA. In *Information Security*, pages 228–243. Springer, 2008.

[41] Yao-Dong Zhao and Wen-Feng Qi. Small private-exponent attack on RSA with primes sharing bits. In *Information Security*, pages 221–229. Springer, 2007.

[42] Hung-Min Sun, Mu-En Wu, Huaxiong Wang, and Jian Guo. On the improvement of the BDF attack on LSBS-RSA. In *Information Security and Privacy*, pages 84–97. Springer, 2008.

[43] Hung-Min Sun, Mu-En Wu, Ron Steinfeld, Jian Guo, and Huaxiong Wang. Cryptanalysis of short exponent RSA with primes sharing least significant bits. In *Cryptology and Network Security*, pages 49–63. Springer, 2008.

[44] M Jason Hinek. *On the security of some variants of RSA*. PhD thesis, 2007.

[45] Chien-Yuan Chen, Cheng-Yuan Ku, and David C Yen. Cryptanalysis of large RSA exponent by using the LLL algorithm. *Applied mathematics and computation*, 169(1):516–525, 2005.

[46] Abderrahmane Nitaj and Mohamed Ould Douh. A new attack on RSA with a composed decryption exponent. *IACR Cryptology ePrint Archive*, 2013: 846, 2013.

[47] Johannes Blömer and Alexander May. New partial key exposure attacks on RSA. In *Advances in Cryptology-CRYPTO 2003*, pages 27–43. Springer, 2003.

[48] Matthias Ernst, Ellen Jochemsz, Alexander May, and Benne De Weger. Partial key exposure attacks on RSA up to full size exponents. In *Advances in Cryptology–EUROCRYPT 2005*, pages 371–386. Springer, 2005.

[49] Marc Joye and Tancrede Lepoint. Partial key exposure on RSA with private exponents larger than N. In *Information Security Practice and Experience*, pages 369–380. Springer, 2012.

[50] Abderrahmane Nitaj. An attack on RSA using LSBs of multiples of the prime factors. In *AFRICACRYPT*, pages 297–310. Springer, 2013.

[51] Ronald L Rivest and Adi Shamir. Efficient factoring based on partial information. In *Advances in Cryptology–EUROCRYPT85*, pages 31–34. Springer, 1986.

[52] Mathias Herrmann and Alexander May. Solving linear equations modulo divisors: On factoring given any bits. In *Advances in Cryptology–ASIACRYPT 2008*, pages 406–424. Springer, 2008.

[53] Dan Boneh and Hovav Shacham. Fast variants of RSA. *CryptoBytes*, 5(1): 1–9, 2002.

[54] J-J Quisquater and Chantal Couvreur. Fast decipherment algorithm for RSA public-key cryptosystem. *Electronics letters*, 18(21):905–907, 1982.

[55] Amos Fiat. Batch RSA. In *Advances in Cryptology–CRYPTO'89 Proceedings*, pages 175–185. Springer, 1990.

[56] Martin E Hellman and Ralph C Merkle. Public key cryptographic apparatus and method, 1980. US Patent 4,218,582.

[57] Tsuyoshi Takagi. Fast RSA-type cryptosystem modulo $p^k q$. In *Advances in Cryptology–CRYPTO'98*, pages 318–326. Springer, 1998.

[58] Hung-Min Sun, Mu-En Wu, M Jason Hinek, Cheng-Ta Yang, and Vincent S Tseng. Trading decryption for speeding encryption in rebalanced RSA. *Journal of Systems and Software*, 82(9):1503–1512, 2009.

[59] Steven D Galbraith, Chris Heneghan, and James F McKee. Tunable balancing of RSA. In *Information Security and Privacy*, pages 280–292. Springer, 2005.

[60] Arjen K Lenstra and Benjamin MM De Weger. Twin RSA. In *Progress in Cryptology–Mycrypt 2005*, pages 222–228. Springer, 2005.

[61] Hung-Min Sun, Mu-En Wu, Wei-Chi Ting, and M Jason Hinek. Dual RSA and its security analysis. *Information Theory, IEEE Transactions on*, 53(8): 2922–2933, 2007.

[62] Daniel Bleichenbacher and Alexander May. New attacks on RSA with small secret CRT-exponents. In *Public Key Cryptography–PKC 2006*, pages 1–13. Springer, 2006.

[63] Guopei Qiao and Kwok-Yan Lam. RSA signature algorithm for microcontroller implementation. In *Smart Card Research and Applications*, pages 353–356. Springer, 2000.

[64] Ellen Jochemsz and Alexander May. A polynomial time attack on RSA with private CRT-exponents smaller than $N^{0.073}$. In *Advances in Cryptology-CRYPTO 2007*, pages 395–411. Springer, 2007.

[65] Bao-jun Gu, Yi Zhou, and Wei-nong Wang. Batch RSA signature scheme. *Journal of Shanghai Jiaotong University (Science)*, 14:290–292, 2009.

[66] Yun-Fei Li, Qing Liu, Lin Hao, and Bao-Lin Zhou. Efficient variant of RSA cryptosystem. *Jisuanji Yingyong/ Journal of Computer Applications*, 30(9): 2393–2397, 2010.

[67] Qing Liu, Yun-Fei Li, Bao-Lin Zhou, and Hua Peng. Research of batch RSA based on multi-prime. *Jisuanji Yingyong Yanjiu*, 28(2), 2011.

[68] Guang Zhao and Hengbo Li. An efficient variant of the Batch RSA cryptosystem. In *Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering*, pages 947–954. Springer, 2013.

[69] Hovav Shacham and Dan Boneh. Improving SSL handshake performance via batching. In *Topics in Cryptology–CT-RSA 2001*, pages 28–43. Springer, 2001.

[70] Gary L Miller. Riemann's hypothesis and tests for primality. *Journal of computer and system sciences*, 13(3):300–317, 1976.

[71] Compaq Computer Corporation. Cryptography using Compaq MultiPrime technology in a parallel processing environment, 2000.

[72] M Jason Hinek, Mo King Low, and Edlyn Teske. On some attacks on multi-prime RSA. In *Selected areas in Cryptography*, pages 385–404. Springer, 2003.

[73] Mathieu Ciet, François Koeune, Fabien Laguillaumie, and Jean-Jacques Quisquater. Short private exponent attacks on fast variants of RSA. *UCL Crypto Group Technical Report Series CG-2003/4, Universite Catholique de Louvain*, 2003.

[74] M Jason Hinek. On the security of multi-prime RSA. *Journal of Mathematical Cryptology*, 2(2):117–147, 2008.

[75] Hatem M Bahig, Ashraf Bhery, and Dieaa I Nassr. Cryptanalysis of multi-prime RSA with small prime difference. In *Information and Communications Security*, pages 33–44. Springer, 2012.

[76] Dan Boneh, Glenn Durfee, and Nick Howgrave-Graham. Factoring $N = p^r q$ for large r. In *Advances in Cryptology–Crypto99*, pages 326–337. Springer, 1999.

[77] Koji Chida, Shigenori Uchiyama, and Taiichi Saito. A new factoring method of integers $N = p^r q$ for large r. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 85(5):1050–1053, 2002.

[78] Alexander May. Secret exponent attacks on RSA-type schemes with moduli $N = p^r q$. In *Public Key Cryptography–PKC 2004*, pages 218–230. Springer, 2004.

[79] Kouichi Itoh, Noboru Kunihiro, and Kaoru Kurosawa. Small secret key attack on a variant of RSA (due to Takagi). In *Topics in Cryptology–CT-RSA 2008*, pages 387–406. Springer, 2008.

[80] Alexander May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In *Advances in Cryptology - CRYPTO 2004*, pages 213–219. Springer, 2004.

[81] Santanu Sarkar. Small secret exponent attack on RSA variant with modulus $N = p^r q$. *Designs, Codes and Cryptography*, pages 1–10, 2014.

[82] Yao Lu, Rui Zhang, and Dongdai Lin. Factoring Multi-power RSA modulus $N = p^r q$ with partial known bits. In *Information Security and Privacy*, pages 57–71. Springer, 2013.

[83] Cesar Alison Monteiro Paixao. An efficient variant of the RSA cryptosystem. *IACR Cryptology ePrint Archive*, 2003:159, 2003.

[84] Hung-Min Sun and Mu-En Wu. Design of rebalanced RSA-CRT for fast encryption. In *Proceedings of Information Security Conference*, pages 16–27, 2005.

[85] Alexander May. Cryptanalysis of unbalanced RSA with small CRT-exponent. In *CRYPTO*, volume 2442, pages 242–256. Springer, 2002.

[86] Scott A Vanstone and Robert J Zuccherato. Short RSA keys and their generation. *Journal of Cryptology*, 8(2):101–114, 1995.

[87] Scott A Vanstone and Robert J Zuccherato. Using four-prime RSA in which some of the bits are specified. *Electronics Letters*, 30(25):2118–2119, 1994.

[88] Arjen K Lenstra. Generating RSA moduli with a predetermined portion. In *Advances in Cryptology–Asiacrypt98*, pages 1–10. Springer, 1998.

[89] Igor E Shparlinski. On RSA moduli with prescribed bit patterns. *Designs, Codes and Cryptography*, 39(1):113–122, 2006.

[90] Naoki Kanayama and Shigenori Uchiyama. The Vanstone-Zuccherato schemes revisited. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 90(12):2903–2907, 2007.

[91] Marc Joye. RSA moduli with a predetermined portion: Techniques and applications. In *Information Security Practice and Experience*, pages 116–130. Springer, 2008.

[92] Xianmeng Meng. On RSA moduli with half of the bits prescribed. *Journal of Number Theory*, 133(1):105–109, 2013.

[93] Santanu Sarkar and Subhamoy Maitra. Cryptanalytic results on 'Dual CRT' and 'Common Prime' RSA. *Designs, codes and cryptography*, 66(1-3):157–174, 2013.

[94] Adi Shamir. RSA for paranoids. *CryptoBytes*, 1(3):1–4, 1995.

[95] Hung-Min Sun, Wu-Chuan Yang, and Chi-Sung Laih. On the design of RSA with short secret exponent. In *Advances in Cryptology–ASIACRYPT99*, pages 150–164. Springer, 1999.

[96] Glenn Durfee and Phong Q Nguyen. Cryptanalysis of the RSA schemes with short secret exponent. In *Advances in Cryptology–ASIACRYPT 2000*, pages 14–29. Springer, 2000.

[97] M Jason Hinek. Another look at small RSA exponents. In *Topics in Cryptology–CT-RSA 2006*, pages 82–98. Springer, 2006.

[98] James McKee and Richard Pinch. Further attacks on server-aided RSA cryptosystems. *to appear*, 1998.

[99] Subhamoy Maitra and Santanu Sarkar. Efficient CRT-RSA decryption for small encryption exponents. In *Topics in Cryptology-CT-RSA 2010*, pages 26–40. Springer, 2010.

[100] Santanu Sarkar and Subhamoy Maitra. More on correcting errors in RSA private keys: Breaking CRT-RSA with low weight decryption exponents. *IACR Cryptology ePrint Archive*, 2012:106, 2012.

[101] Victor Shoup. Ntl: A library for doing number theory, 2001.

[102] David Pointcheval. New public key cryptosystems based on the dependent-RSA problems. In *Advances in Cryptology–EUROCRYPT'99*, pages 239–254. Springer, 1999.