August 2012

Master's Degree Thesis

On the Method of Inverse Modulo 2^k for PKC Implementations

Graduate School of Chosun University

Department of Information and Communications Engineering

Anish Bahadur Amatya

On the Method of Inverse Modulo 2^k for PKC Implementations

공개키 암호 구현을 위한 모듈로 2^k의 역원 연산 방법

August 24, 2012

Graduate School of Chosun University

Department of Information and Communications Engineering

Anish Bahadur Amatya

On the Method of Inverse Modulo 2^k for PKC Implementations

Advisor: Prof. Young-Sik Kim

This thesis is submitted to Chosun University in partial fulfillment of the requirements for a Master's degree in engineering

April, 2012

Graduate School of Chosun University

Department of Information and Communications Engineering

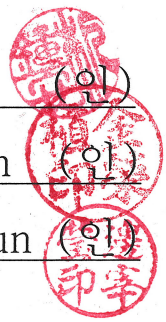
Anish Bahadur Amatya

Anish Bahadur Amatya's Master's Degree Thesis Approval

Committee Chairperson Prof. Jong-An Park (인)

Committee Member Prof. Young-Sik Kim (인)

Committee Member Prof. Jae-Young Pyun (인)



May 2012

Graduate School of Chosun Univesity

Table of Contents

List of Tables

List of Figures

초 록

공개키 암호 구현을 위한 모듈로 2^k 의 역원 연산 방법

Anish Bahadur Amatya
지도 교수: 김영식, 교수, Ph. D.
정보통신공학과, 대학원 조선 대학교

오늘날 공개키 암호 시스템은 정보 보호 프로토콜을 구현함에 있어서 부인방지 기능과 같은 필수적인 기능을 제공한다. 그러나 공개키 암호 시스템은 비밀키 시스템에 비해서 더 긴 비밀정보의 길이를 갖고 있을 뿐만 아니라, 암호화 및 복호화를 수행하는데 있어 더 많은 연산량을 필요로 한다는 문제를 갖고 있다. 일반적으로 공개키 암호 시스템은 모듈러 덧셈, 뺄셈, 곱셈, 및 역원 연산을 일정 순서에 따라 수행하게 된다. 이러한 연산을 효율적으로 수행하기 위해서 Montgomery 모듈러 곱셈이나 Jebelean 의 정확한 나눗셈 기법과 같은 것들이 사용된 바 있다. 그 중에서 모듈러 역원은 입력의 크기가 클 때 많은 시간을 소모하고, 계산이 복잡한 과정으로 잘 알려져 있다. Arazi 와 Qi 는 재귀적인 방식으로 역원을 구할 수 있도록 하나의 변수를 반으로 분할하여 2 의 거듭 제곱 형태의 모듈러스를 갖는 역원 연산을 효율적으로 수행할 수 있는 알고리즘을 제안한 바 있다.

이 논문에서는 Arazi 와 Qi 의 방법을 변경하여 특별한 경우에 더욱 효율적인 연산이 수행한 단순화된 알고리즘을 제안한다. 연산에 사용되는 숫자들이 특별한 구조를 갖는 경우 연산 알고리즘은 보다 단순화될 수 있고, 그 결과 재귀적 과정이 제거되어 고속으로 효율적인 연산을 수행할 수 있다는 사실을 밝혀 내었다. 시뮬레이션 결과에 따르면 새롭게 제안한 알고리즘은 2 의 거듭 제곱 형태의 모듈러스를 갖는 역원 연산을 기존의 Arazi 와 Qi 가 제안한 방식뿐만 아니라, Dusse 및 Kaliski 의 연산 법, 직접적인 연산 법, 그리고 확장된 유클리드 알고리즘 보다 더 빠르게 연산할 수 있다는 사실을 확인할 수가 있었다. 그러나 이러한 고속의 성능은 입력되는 숫자가 특별한 구조를 취한 경우에만 달성될 수 있다는 제약을 갖는다. 하지만 암호학적 연산의 경우에는 이 논문에서 가정한 특별한 구조를 갖는 숫자를 이용하는 경우가 많이 있기 때문에, 이 논문에 따르면 그러한 연산의 경우 보다 효율적인 고속 연산이 가능한 것을 확인할 수가 있다.

ABSTRACT

On the Method of Inverse Modulo 2^k for PKC Implementations

Anish Bahadur Amatya
Advisor: Prof. Young-Sik Kim, Ph.D.
Department of Information and Communications Engineering,
Graduate School of Chosun University

Public key crypto-systems (PKCs) offer powerful and convenient methods for implementation of information security, since we do not need to disseminate a common secret key before starting communication. However, PKCs require more computational resources and also need to have a key of larger size for the system to be as secure as secret key crypto-systems. Most PKCs are based on modular operations such as modular addition, subtraction, multiplication and inversion. Among them, modular inversion is a relatively more time consuming and computationally complicated process, especially when the size of the input is large. A special case is inverse modulo power of 2 which is needed in the algorithms required for PKC implementations, such as Montgomery modular multiplication and Jebelean's exact division method. Arazi and Qi have proposed an efficient algorithm for calculation of inversion modulo power of 2, which recursively divides a number into halves in order to find the required inverse.

This thesis gives a detail description of the method as well as proposes a simplification. A special structure of numbers is introduced, which helps to eliminate the recursion process,

hence making the algorithm simpler, execute faster and computationally more efficient. It shows that the calculation of inverse modulo a power of 2 can be done faster than the method proposed by Arazi and Qi, as well as faster than popular methods such as Dusse and Kaliski's method, straight forward method and extended Euclidean method. However, in their binary representation the input numbers should confirm to a special structure.

I. Introduction

Communication is a very important aspect of life. It is the process by which we can send information from one person or place to another. It can be done in many different ways. A simple practical example is direct communication, such as talking in front the recipient, or telling them the required information through a telephone. Another way is written communication, for example, writing a letter or sending an email. In any communication system, information is produced at the source and changed into a form suitable for transmission through the channel. At the receiver, the signal is recovered through appropriate processing [1]. In its most simplified form, a communication system consists of a source, a communication channel and a destination.

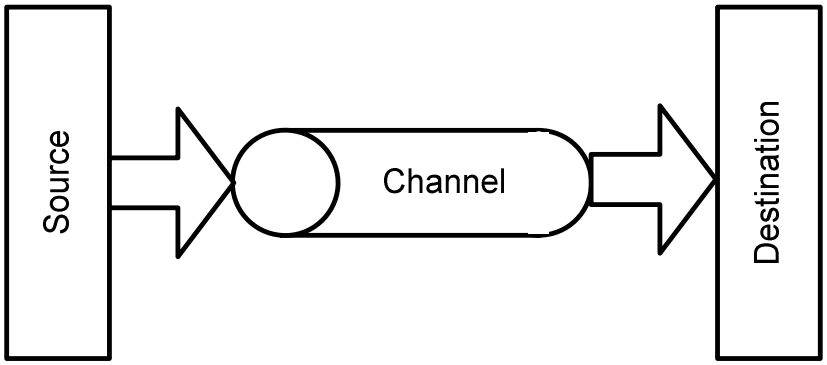


Fig. 1.1 Communication system block diagram

Fig. 1.1 shows the most basic blocks in a communication system. However, in most of the cases the notion of secure communication is also equally important. The communication channel not only provides a chance for loss or damage of information, but also a possible theft of information. A channel where there is a possibility of loss of information, damage to the data being transferred or even simply tapping of information, is referred to as an insecure channel while a channel with no such adversary is called a secure channel [2]. We must note that a secure channel is an ideal channel and its practical realization may not be feasible. In most of the cases the channel may have interferences which may be due to an inherent

property of the channel [1] as well as introduced artificially. Hence a way to protect our data from such interferences for reliable data transfer is mandatory. Information security, cryptography in particular, provides a way to protect the transmitted data from such introduced interferences, as well as attempts to hide the information being transferred from unauthorized entities that may be present in the channel. The basic service it provides is the ability to send information between participants in a way that prevents others from reading it [3].

A. Thesis Motivation and Overview

Cryptography involves the process of encryption, which is the process of converting information in readable form into data in incomprehensible form. Although the incomprehensible data may be read, it should be very difficult (and ideally impossible) for an unauthorized entity to be able to re-obtain the initial readable information from it. Moreover it should be relatively easy for the intended recipients of the message to be able to decipher the incomprehensible data into readable information. This is achieved by passing the information to be transferred through an encryption routine. It changes the input information consisting of alphabets from a finite set into a sequence of alphabets in another finite set (or usually, the same set). The algorithm used for this conversion is publicly known. However, the values that a given input converts to, is determined by another piece of information associated with the algorithm, known as a key. This can also be expressed mathematically. Let A denote a finite set called the alphabet of definition. An example would be $\mathrm{A} = \{0,1\}$, which is the binary alphabet. M denotes a set called message space which consists of strings of symbols from the alphabet of definition. An element of this set is called plaintext. C denotes a set called the ciphertext space. It consists of strings of symbols from the alphabet of definition, which may differ from the alphabet of definition for M. An element of this set is called a ciphertext. K denotes a set called key space and an element of this is called a key.

Each element $e \in K$ determines a one-to-one transformation from M to C, denoted by E_e, called an encryption function or an encryption transformation. For each $d \in K$, D_d

denotes a bijection from C to M. D_d is called a decryption function or decryption transformation. The process of applying the transformation E_e to a message $m \in M$ is called encryption and the application of the transformation D_d to a ciphertext c to obtain m is called decryption [4]. An encryption scheme consists of $\{E_e : e \in \mathrm{K}\}$ of encryption transformations and a corresponding set $\{D_d : d \in \mathrm{K}\}$ of decryption transformations with the property that for each $e \in K$ there is a unique key $d \in K$ such that $D_d = E_e^{-1}$; that is, $D_d(E_e(m)) = m$ for all $m \in M$. An encryption scheme is sometimes referred to as a cipher. The keys e and d in the preceding definition are referred to as a key pair and sometimes denoted by (e, d) [4]. If the key used in changing comprehensible data into incomprehensible form (encryption) and vice versa (decryption) is the same, then the method is known as Symmetric or Private Key Cryptosystem. If the keys used are different, then the method is known as Asymmetric Key Cryptosystem. Normally the key used for decryption is kept secret by the receiver while the key used for encryption is publicly known. Hence this system is also known as Public Key Cryptosystem.

Public Key Cryptosystem or PKC obviously provides ease of use as the need to securely transfer a secret key to the communicating parties is not necessary. However, to achieve a level of security same as that or higher than Symmetric Key Cryptosystems, PKC requires keys of much larger bit length. The larger the bit length of the key, the more secure the algorithm is. But the higher bit length also contributes to significant increase in the algorithm execution time. If a method of reducing this time of execution can be developed, it will make the practical implementation and deployment of PKCs highly efficient. This thesis attempts to provide some background information on PKC, the different algorithms popular in PKC implementation and how the speedup of the implementation of one such algorithm may be obtained, on the basis of certain mathematical conditions being met.

B. Research Objectives

Some disadvantages associated with PKCs are the complexity of the algorithms, large size keys and hence the longer execution time. These are also the reasons why they are mainly used for key transfer or digital signatures instead of actual data encryption. The most commonly used PKC systems utilize the RSA encryption scheme based on the hardness of factoring a large number with large prime factors. Other methods such as Diffie-Hellman Key exchange, based on the discrete logarithm problem are also available. The common characteristic between the various algorithms is that they require operations based on modular mathematics. Modular addition and subtraction algorithms are easy to understand and implement and are considered easier to implement. Algorithms for operations like division, multiplication and inversion are much harder and significantly more time consuming, especially when large size numbers are used.

Many algorithms have been developed that help to increase the speed of execution of such modular operations, mainly modular multiplication, division and inversion. However, as the bit length of the input numbers keep on increasing for higher level of security, methods to further increase the speed of execution must also be devised.

This study mainly focuses on improving the implementation of PKCs by increasing the speed of calculation of the inverse of a number modulo a power of 2. Calculating inverse is considered one of the most time-consuming operations in modular arithmetic and increasing the speed efficiency of its execution will greatly help in its better implementation.

C. Thesis Contribution

This thesis provides a simplification on a previous algorithm developed for the calculation of inverse modulo power of 2. This simplification results in a higher speed of execution. Many algorithms for the calculation of modular inverses have been developed in [5]

- [9]. As already mentioned calculating modular inverses is one of the most time-consuming of all modular operations. Since computers work on binary numbers, if the modulus is a power of 2, it is possible to modify the general methods used for finding modular inverse [4], [5] to construct a more efficient one. Hence special algorithms designed to efficiently calculate inverse modulo power of 2 have also been developed. Still, the time required for the execution of these algorithms is also significantly great for larger sized numbers. Hence Arazi and Qi in [10] suggested an alternative method for calculating inverse modulo a power of 2. This study intends to take it a step further. It makes certain assumptions about the input numbers and proposes a simplification on the original algorithm given in [10].

1. Simplified Algorithm:

A simplified version of the algorithm proposed in [10] is given, along with certain assumptions about the numbers being operated on. These assumptions are based on another paper dealing with improving the execution speed of Montgomery multiplication method implementation. As long as the assumptions for the input numbers are valid, the simplified algorithm has a faster execution time than the original unmodified algorithm for the same input.

2. Result of Comparison:

Code was written in C implementing the original algorithm by Arazi and Qi, Dusse and Kaliski's method [11], straight forward method [10], modification of extended Euclidean algorithm for inverse modulo power of 2 [10] as well as the simplified version Arazi and Qi's algorithm proposed here. This thesis provides the result of comparison between these various methods for different values of input numbers as well as for inputs of different bit lengths. It shows that the simplified algorithm does provide an improvement over the original algorithm as well as the other popular methods, especially when the bit length of the input numbers increases.

D. Thesis Organization

This thesis is organized into chapters as follows. Chapter II presents an overview of information security, especially public key cryptography and the various modular arithmetic algorithms used. It also describes in detail the algorithms used for inversion modulo power of 2 which was the focus of this study. Chapter III describes Arazi and Qi's method as well as introduces the special structure of numbers. The chapter then describes the simplification that can be done on the modular inverse algorithm studied, as well as shows the results of comparison between the simplified algorithm, the original algorithm as well as the other popular algorithms for inversion modulo 2. Chapter IV summarizes and concludes the thesis, with some suggestions for future modifications.

II. Background

This chapter is devoted to the background necessary for discussing the work in this thesis. Section A provides an overview of the basics of information security and cryptography in general. Section B introduces some of the mathematical algorithms needed for PKC implementations that are relevant to this thesis. It first describes the Montgomery reduction algorithm needed to speed up modular multiplication. This is followed by Dusse and Kaliski's algorithms and the straight forward method. The chapter ends with a description of the modified extended Euclidean method for finding inverse modulo the power of 2.

A. Overview of Information Security

As mentioned in Chapter I, the ideal communication system depicted in Fig. 1.1 is not always practically feasible. The channel through which data needs to be transferred consists of various sources of noise and interference. Some of these are present due to the properties and characteristics of the channel itself, such as signal attenuation and introduction of random noise. The methods described in this thesis and cryptography in general, however, deals with protecting the data from other types of intrusions that may be present on the channel, such as some unauthorized entities trying to read the information being transferred or trying to alter it in some way. In the terminology of cryptography, the communicating parties are usually known as Alice and Bob while the intruder is knows as Eve. When Alice and Bob are trying to communicate through a channel as shown in Fig. 1.1, Eve may be present on the channel altering the information being transferred, without the knowledge of either Alice or Bob. If this is the case, the message that the sender wants the receiver to obtain is not the one that he gets. It may also be the case that Eve is continuously watching the channel, not altering it in any way, but observing the information that is being transferred between Alice and Bob. In most of the cases this is also an undesirable situation. For example, if the communication between Alice and Bob was a bank transaction, it would certainly not be desirable for a third

party to know about the details. Hence the communication system model shown in Fig. 1.1 is incomplete. It can be better described by the block diagram shown below [12]:

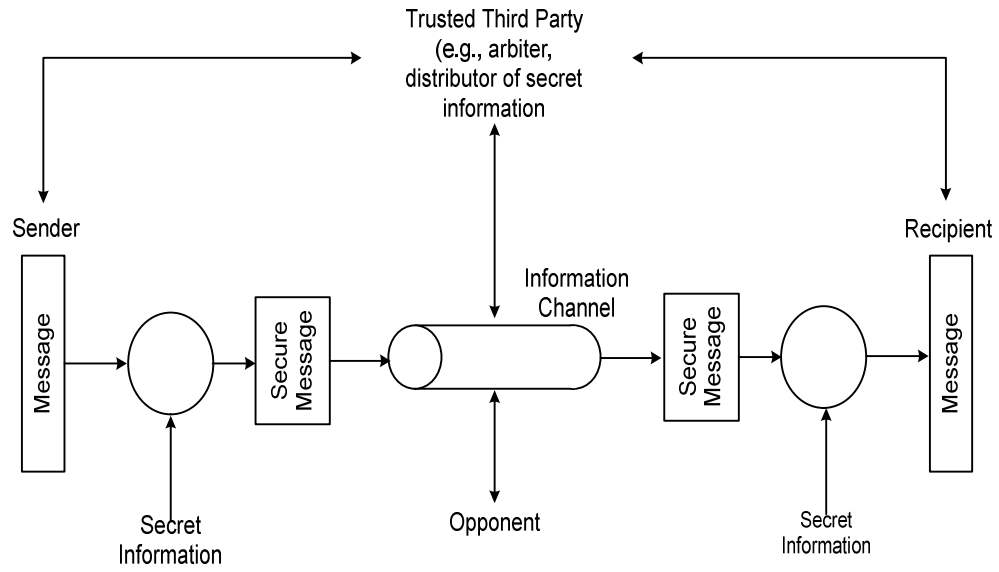


Fig. 2.1 Model for network security

As can be seen from Fig. 2.1, this new communication system model acknowledges the presence of an opponent in the information channel. However it also provides a way of preventing this opponent from tampering with the transferred data. We can see that a trusted third party is overseeing the sender and the receiver as well as the communication channel. However the main focus of this thesis lies in the blocks between the sender and the channel as well as the channel and the recipient. These are the blocks that, with the help of some extra information, do the necessary transformations on the message. This is also the main point behind cryptography. The process of converting legible information from sender, using a piece of information called key, into illegible secure message or cipher text to be transmitted through the information channel is known as encryption. The opposite process of converting the cipher text obtained from the information channel into legible message for the receiver using a key is known as decryption.

Before going into the details of security, the various ways in which the communication between Alice and Bob may be compromised is given here. They represent the manners in which an eavesdropper, Eve, may disrupt, alter, or simply find out, in an unauthorized manner, the information being transferred between Alice and Bob.

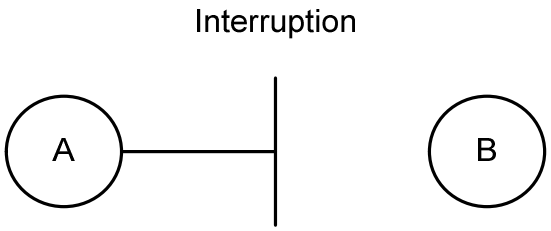


Fig. 2.2 Interruption of communication

a) Interruption: Interruption is the threat on availability. Fig. 2.2 shows the concept of interruption of transmitted message. The attacker may simply block the communication channel, or in some other way make it infeasible for the receiver to obtain the data transmitted by the sender.

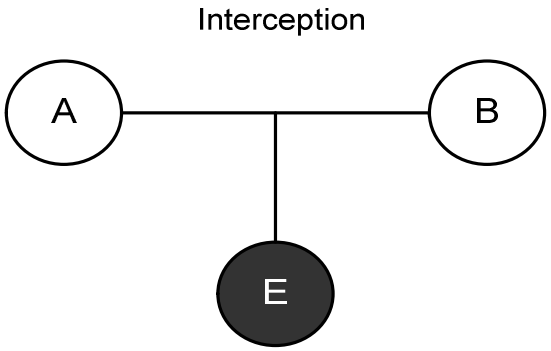


Fig. 2.3 Interception of data

b) Interception: Fig. 2.3 represents interception of transmitted data, which is the threat on confidentiality. In this kind of attack the attacker will allow the transfer of data from Alice to Bob or vice versa, but will be monitoring the data that is being transferred, usually without their knowledge.

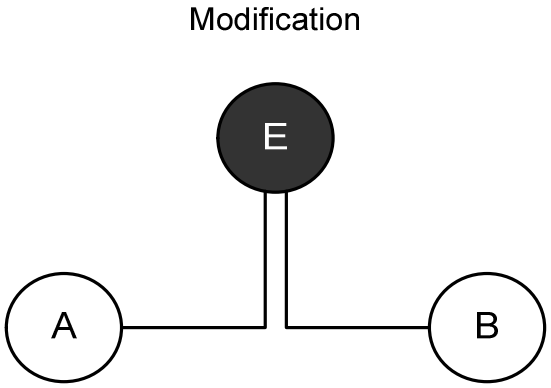


Fig. 2.4 Modification of data

c) Modification: Fig. 2.4 shows how the attacker is compromising the integrity of data. In this case the attacker obtains data from the sender, modifies it and then transmits it to the receiver, usually without the knowledge of either party. An implementation of this attack would be the man-in-the-middle attack done against public key systems [2], [3], [12], [13]. In an ideal attack the sender and receiver will be completely oblivious to the presence of the attacker and will not be able to tell if their communication has been compromised .

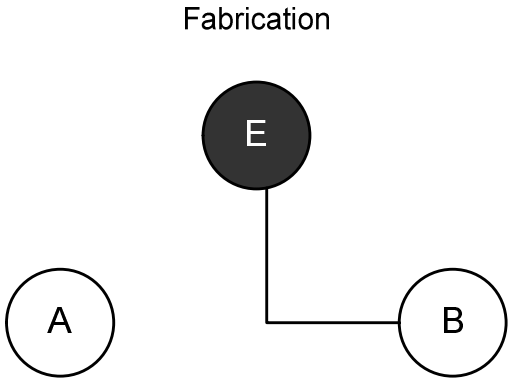


Fig. 2.5 Fabrication of data

d) **Fabrication**: Fabrication corresponds to a threat on authenticity of data being transferred. Fig. 2.5 represents such an attack. As can be seen, an attacker is impersonating someone else. An adversary replacing unauthenticated public keys with his own in a public key infrastructure can be an example of such an attack [13]. The attacker sends data to the receiver as if they were originating from the sender. In an ideal attack the receiver cannot tell whether the received information is legitimate or illegitimate.

In a more general sense, any kind of attack may be categorized into one of two types: passive attack and active attack [12]. In a passive attack the attacker attempts to learn or make use of information from the system, but does not affect the system resource. Side channel attacks may be categorized as a type of passive attack. The goal of the attacker is to obtain the information that is being transmitted. A passive attack may be done to release the contents of the message, which is clearly not desirable for confidential information. Alternatively, passive attack may also be done to gather data for traffic analysis. It is possible to mask the actual data being transferred through the channel, so that even if the data could be captured, it would not be possible to extract meaningful information from the data. This can be achieved by encryption which will be further discussed later. Even with such a protection in place, an attacker might still be able to observe the pattern of the masked messages and through their careful analysis determine the location and identity of communicating hosts as well as observe the frequency and length of messages being exchanged. Such information may be useful in guessing the nature of the communication taking place. Passive attacks are mainly attacks on the encryption scheme, done to systematically recover plaintext from ciphertext or even attempt to deduce the decryption key [4].

Active attacks on the other hand involve some kind of modification of data stream or a creation of a false stream and can be subdivided into four categories [12].

a) **Masquerade**: In such an attack the attacker pretends to be either the sender or the receiver. It is usually done to perform another form of active attack. For example, authentication sequences can be captured and replayed after a valid authentication sequence has taken place, thus enabling an entity with few privileges to impersonate another one with higher privileges.

b) **Replay**: This involves the passive capture of a data unit and its subsequent retransmission, to produce an unauthorized effect.

c) **Modification of messages**: It implies that some portion of a legitimate message is altered, or that messages are delayed or reordered, to produce an unauthorized effect. For example, a message meaning "Allow John Smith to read confidential file accounts" is modified to mean "Allow Fred Brown to read confidential file accounts."

d) **Denial of service**: The denial of service prevents or inhibits the normal use or management of communications facilities. This attack may have a specific target; for example, an entity may suppress all messages directed to a particular destination (e.g., the security audit service).Another form of service denial is the disruption of an entire network, either by disabling the network or by overloading it with messages so as to degrade performance for other legitimate users.

We can see that active attacks are opposite in nature to passive attacks. Although passive attacks are difficult to detect, methods can be implemented to prevent them. On the other hand, it is very difficult to completely prevent active attacks since there are wide variety of potential physical, software, and network vulnerabilities. Instead, we try to detect active attacks and recover from any disruption or delays caused by them [12]. If the detection has a deterrent effect, it can also contribute to prevention of the attack in the first place. Mostly protection from all these attacks is provided using masking of data being transferred and by following certain security protocols. In any case, encryption of the data being transferred is a necessity and plays a very vital role.

As already mentioned encryption basically implies converting the legible data to be transmitted into illegible data. The data to be transferred is called the message text while the converted illegible data is called the ciphertext. The algorithm that does this conversion is known. However, the exact ciphertext that any given message converts to depends on another piece of information called an encryption key. Similarly, the conversion of the illegible data into legible message at the receiver's end is done using the decryption algorithm and the decryption key. If M represents the message to be encrypted, E the encryption algorithm, D

the decryption algorithm, C the ciphertext and K_1 and K_2 the encryption and decryption key respectively, the process of encryption and decryption can be represented by the transformations

$$C = E_{k_1}(M) \quad \text{..... (1)}$$

$$M = D_{K_2}(C) \quad \text{..... (2)}$$

respectively [12]. Depending on the value of the keys used, K_1 and K_2, cryptosystems can be divided into two broad categories; symmetric key cryptosystem and asymmetric key cryptosystem.

a) **Symmetric Key Cryptosystem**: In this type of system, the two keys used for encryption and decryption, K_1 and K_2, are same. Hence this system is also known as symmetric key encryption. As already mentioned, since the details of the encryption and decryption algorithm are known, the keys used have to be kept secret. One secret key, for both encryption and decryption, are known to both the sending and receiving parties and have to be shared between them prior to encrypted communication through a secure channel. Examples of symmetric key cryptosystems include classical ciphers such as Ceaser cipher, Vignere cipher and more recent ones such as AES, DES, etc.

b) **Asymmetric Key Cryptosystem**: In this system the encryption key and the decryption key are related but different. Hence this type of system is also known as asymmetric key cryptosystem. Such a concept was introduced by Diffie and Hellman in [14]. One of the keys is used for encryption, however, only the other key can then be used for decryption of the message. Usually the encryption key is known publicly while the decryption key is kept secret, so that only the intended recipient has the key required for decryption. However encryption may be carried out using the secret key, when creating a signature. In any case, if one if the keys is used for encryption, the other will be used for decryption. Also, knowledge of the public key does not make it easier (and ideally is impossible) to determine the corresponding secret key. Most

implementations of these methods base their security on the hardness of some particular mathematical problems. Some of the more popular examples are RSA and Diffie-Hellman Key exchange [15].

In its most intuitive form, any form of encryption consists of transforming some information, consisting of symbols from a finite set of symbols (the message text) into data consisting of symbols from another (but usually the same) set of symbols (the cipher text). Hence this gives rise to the concept of modular arithmetic when creating mathematical descriptions of cryptosystems. The most basic idea of modular arithmetic is that it works on a finite set of numbers. Hence, unlike in classical arithmetic where we can allow the results of our operation to range from negative infinity to positive infinity with each number being distinct from any other, the numbers in modular arithmetic repeat after a certain count, depending on the modulus used. A little more about modular arithmetic will be discussed here. The mod operator is an important operation in modular arithmetic. The operation $X \bmod Y$ represents the remainder when X is divided by Y. The congruence relation $A \equiv B \bmod N$ read as A congruent to B modulo N, implies that the number A is the summation of some multiple of N and B, that is, $A = k \cdot N + B$. It also implies that $(A \bmod N) = (B \bmod N)$. It should be noted however that all the parameters to be considered here are integers. Modular operations have certain properties that make them distinct from their corresponding regular arithmetic counterparts. Some of the more important and basic properties of modular mathematics are as follows [12]:

1. $a \equiv b \bmod n$ if $n \mid (a - b)$
2. $a \equiv b \bmod n$ implies $b \equiv a \bmod n$
3. $a \equiv b \bmod n$ and $b \equiv c \bmod n$ implies $a \equiv c \bmod n$

These were the properties of the modulo (mod) operator. By definition the mod operator maps all integers in to the set of integers $\{0,1,2,...,(n-1)\}$. It is also possible to perform arithmetic operations such as addition and multiplication within the confines of this set. This is known as modular arithmetic. Some important properties of modular arithmetic are:

1. $[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$
2. $[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$
3. $[(a \bmod n) * (b \bmod n)] \bmod n = (a * b) \bmod n$

for any integers a, b, and n.

Also, if we have the integers a, b, and c working modulo the integer n, then if $(a+b) \equiv (a+c) \bmod n$, it implies $b \equiv c \bmod n$. However, if we are concerned with multiplication, if the expression $(a \cdot b) \equiv (a \cdot c) \bmod n$ is true then $b \equiv c \bmod n$ if and only if a is relatively prime to n. The two numbers a and n are said to be relatively prime or coprime to each other, when the greatest common divisor of the two numbers is 1, that is, if the largest number that can divide both of them is 1.

By its definition cryptography can include any mechanism that can be used to encrypt, decrypt, sign or authenticate data. This can include mechanical, electromechanical, electronic or quantum–mechanical systems. However in practice today cryptography generally implies electronic systems, specifically hardware and software systems that can be implemented in a computer, as cryptographic algorithms have become more complex and more demanding. This thesis will concentrate on a particular discipline of cryptography, more specifically on public key cryptosystems and discusses on how a speedup on their implementation may be achieved by focusing on a particular widely used operation, the inverse modulo power of 2.

B. Algorithms Involved in PKC

Asymmetric key cryptosystems are relatively more convenient to use than symmetric key cryptosystems, as they do not need the sharing of a secret key through a secure channel. Encryption and decryption are carried out by different keys, and only one of the keys needs to be kept secret while the other one can be made public. It is not feasible for anybody to derive the secret key from the publicly available one. The list of public keys must be kept safe so that they cannot be tampered with. Since the public key, if used for encryption, cannot be used for

decryption, only the intended recipient who holds the secret key can decrypt the message. An actual implementation of a PKC consists of different types of modular arithmetic operations. In a typical PKC implementation, for example, in the RSA cryptosystem, the hardness of determining the private key from the public key arises from the hardness of factorizing large integer numbers and for actual encryption and decryption the process requires multiple modular multiplications to be carried out in the form of modular exponentiation.

Multiplication however, is one of the most time consuming of modular operations especially when the size of the numbers under consideration is very large [16]. To reduce the amount of time needed for its execution, various types of modifications to the schoolbook multiplication algorithms [17] - [19] as well as completely different algorithms to be used in conjunction with multiplication algorithms, such as the Montgomery modular reduction algorithm have been developed [16], [18]. However, such methods have further given rise to the need to develop algorithms for other time consuming operations such as modular division and modular inverse. Although this study focuses on cryptography, these algorithms find uses in other fields too [10], for example, multiplication algorithms are needed in signal processing and coding theory as well [17]. Hence methods that help to increase the speed of execution of these operations will be helpful in many different ways.

Firstly the Montgomery multiplication method will be briefly discussed [4]. Montgomery reduction is a technique which allows efficient efficient implementation of modular multiplication without explicitly carrying out the classical modular reduction step [4], [20]. If m is a positive integer, R and T are integers such that $R > m$, $\gcd(m, R) = 1$ and $0 \le T \le mR$, the value of $TR^{-1} \bmod m$ can be determined without actually using the product or division operation to determine the remainder. $TR^{-1} \bmod m$ is the Montgomery reduction of T modulo m with respect to R. Using a suitable choice of R, Montgomery reduction can be efficiently computed. Let x and y be integers such that $0 \le x, y < m$. Let $\tilde{x} = xR \bmod m$ and $\tilde{y} = yR \bmod m$. The Montgomery reduction of $\tilde{x}\tilde{y}$ is $\tilde{x}\tilde{y}R^{-1} \bmod m = xyR \bmod m$. It should be noted that this fact can be used to provide a more efficient method for modular

exponentiation. If m is represented as a base b integer of length n, then a typical choice for R is b^n. He condition $R > m$ is obviously satisfied, however, $\gcd(R, m) = 1$ will only hold if $\gcd(B, m) = 1$. Thus this choice of R is not possible for all moduli. For those moduli of practical interest (for example, RSA moduli), m will be odd; then b can be a power of 2 and $R = b^n$ will suffice. Given integers m and R where $\gcd(R, m) = 1$, let $m' = -m^{-1} \bmod R$ and let T be any integer such that $0 \le T < mR$. If $U = Tm' \bmod R$ then $(T + Um)/R$ is an integer and $(T + Um)/R \equiv TR^{-1} (\bmod m)$. This fact can be mathematically proven as follows. We know that $T + Um \equiv T (\bmod m)$ and hence $(T + Um)R^{-1} \equiv TR^{-1} (\bmod m)$. To see that $(T + Um)R^{-1}$ is an integer, we should observe that $U = Tm' + kR$ and $m'm = -1 + lR$ for some integers k and l. It follows that

$$\frac{T + Um}{R}$$

$$= \frac{T + (Tm' + kR)m}{R}$$

$$= \frac{T + T(-1 + lR) + kRm}{R}$$

$$= lT + km \quad \text{..... (3)}$$

Also we should note that $(T + Um)/R$ is an estimate for $TR^{-1} \bmod m$. Since $T < mR$ and $U < R$ we have

$$\frac{T + Um}{R} < \frac{mR + mR}{R} = 2m \quad \text{..... (4)}$$

Thus we can say that one of expressions among either $(T + Um)/R = TR^{-1} \bmod m$ or $(T + Um)/R = (TR^{-1} \bmod m) + m$ is true, that is, the estimate may exceed in value by at most

m. This process of Montgomery reduction can be used with multiple precision multiplication algorithm to compute Montgomery reduction of the product of two integers [4].

In simple terms, Montgomery reduction converts the numbers to be operated upon into integers in the Montgomery domain, where the actual multiplication operation is carried out. This is done by converting each digit of the number under operation into 0 from the LSB towards MSB, by adding another single digit number. This addition in turn changes the value of the next higher digit, so at every iteration the digit to be added so that the next higher digit can be changed to 0 must be calculated. This process continues until enough lower digits have been converted to 0, so that they can be removed by a division by R, or a multiplication by R^{-1}. Hence to make the implementation simpler R is usually chosen to be the base used for representation of the numbers raised to the power number of digits in the modulus in that given base. This is due to the requirement that R must be greater than the modulus. Obviously the final result needs to be converted back from the Montgomery domain. Thus the fact that we need to convert into and out of the Montgomery domain adds to the overhead when using this method to speed up modular multiplication. Due to this reason, this process is more efficient when multiple modular multiplications need to be performed over the same modulus [19]. It may in fact be more inefficient than straightforward methods in case we need to carry out single or few modular multiplications, depending on the size of the numbers.

Calculating the modular inverse of an integer is another operation that is more time consuming among modular arithmetic functions, especially for larger sized numbers. For modular inverse to exist, the number whose inverse is to be calculated needs to be relatively prime to the modulus [12]. Modular inverse can be found using the extended Euclidean algorithm. For implementation in binary computers, an adaptation of the extended Euclidean algorithm using the binary GCD algorithm has also been developed [5]. However these methods were still quite complicated to implement in hardware and software and took relatively more time to execute as well. Hence the Montgomery modular inverse method was developed [6], which has been further modified and built upon by various researchers [8], [13] to provide further ease of implementation as well as more efficiency in execution speed.

However, if we need to calculate the inverse modulo power of 2, other more dedicated methods can be used to perform the operation [10], [11], which are more efficient. It should be noted that it has been possible to design these algorithms more efficiently mostly because computers are designed to work in binary arithmetic. None the less, these algorithms specially tailored to finding inverse modulo power of 2 are better than other general methods that just use a power of 2 number as the modulus, and are the focus of this thesis. The operation of calculating multiplicative inverses modulo power of 2 finds uses in applications such as Montgomery modular multiplication and exact division [7], [10]. Some of the more popular algorithms that had previously been developed to perform this operation will be presented here. They are either more efficient or more intuitive than using the more general algorithms for finding modular inverse by setting the value of the modulus as a power of 2.

Algorithm 1: Dusse and Kaliski's method to calculate $r = b^{-1} \bmod 2^m$

The algorithm for calculating inverse modulo power of 2 using Dusse and Kaliski's method is given below [11], [21].

$y_1 = 1$

for $i = 2$ *to* m **do**

  if $b \cdot y_{i-1} < 2^{i-1} \bmod 2^i$ **then**

    $y_i = y_{i-1}$

  else

    $y_i = y_{i-1} + 2^{i-1}$

  end

end

Fig. 2.6 Dusse and Kaliski's method to calculate inverse modulo power of 2

Fig. 2.6 shows us Dusse and Kaliski's method for calculating the inverse of a number modulo a power of 2. At the end of the iteration, the value of y_m is the desired result $r = b^{-1} \bmod 2^m$. The method determines the value of the inverse number bit by bit starting

from the LSB, keeping in mind the fact that the lower m bits of the product of the input number and its inverse should be 1.

Algorithm 2: Straight forward method

Algorithm 2 is a more straight forward method for calculating inverse modulo power of 2 [10]. It utilizes the fact that b is an odd number and thus has 1 as the least significant bit, and the fact that the product of b and b^{-1} modulo 2^m is 1, so in binary representation of m bits, is of the form $000\ldots01$. Thus by adding selected left shifts of b to an accumulated sum that starts with b, we can always generate a value whose m LSBs are of any given form, including $000\ldots01$. This algorithm is thus executed by sliding b leftwards one bit at a time across an accumulated sum such that the LSB of b generates the bits of the given product. The steps of the algorithm for calculating $r = b^{-1} \bmod 2^m$ are as follows:

$acc = b$

$x = 1$

$y = b$

$res = 1$

for $i = 1$ *to* $n - 1$ **do**

    $y << 1$

    $x << 1$

    if i^{th} bit of $acc = 1$

        $acc \leftarrow acc + y$

        $res \leftarrow res + x$

    end

end

Fig. 2.7 Straight forward method of calculating inverse modulo power of 2

Fig. 2.7 shows the straight forward method of finding the inverse modulo a power of 2 number. Since this method uses, in a way, the definition of modular inverse in the binary

representation, this method is the easiest and most intuitive way to implement this operation. However, it is also the slowest method. In terms of complexity, this method is same as multiplying two m bit numbers. Upon careful observation this method is in fact very similar to the schoolbook method of multiplication of two m bit numbers.

Algorithm 3: Modification of extended Euclidean algorithm

This algorithm is more closely related to the binary GCD version of the extended Euclidean algorithm and it completes in two stages. To calculate $b^{-1} \bmod 2^m$ the algorithm first computes $2^{-m} \bmod b$ and in the next stage interchanges the roles of the two values 2^m and b. The procedure is as given below:

Algorithm 3a: calculating $2^{-m} \bmod b$ by m successive divisions of $2 \bmod b$

$d = 1$

for $i = 1$ *to* m **do**

if d is odd then

$d = d + b$

end

$d = d / 2$

end

The final value of d is $2^{-m} \bmod b$.

Algorithm 3b: recovering $r = b^{-1} \bmod 2^m$ out of $s = 2^{-m} \bmod b$.

$t = s \cdot 2^m$

$u = (t - 1) / b$

$r = s^m - u$

Fig. 2.8 Modified extended Euclidean method to calculate inverse modulo power of 2

Fig. 2.8 is the modified extended Euclidean method for calculating inverse of a number modulo power of 2. As can be seen from the above figure, the process completes in two parts.

The first part calculates the value $2^{-m} \bmod b$. It should be noted that the relation $s = 2^{-m} \bmod b$ implies that $t = s \cdot 2^m = u \cdot b + 1$ and so the value u calculated in the second step of Algorithm 3b, is therefore an integer. Thus, $u \cdot b = s \cdot 2^m - 1$. Taking both sides of the latter relation modulo 2^m yields the congruence $u \cdot b = -1 \bmod 2^m$. From this we can say that $(-u) \cdot b = 1 \bmod 2^m$ and therefore, $-u = r = b^{-1} \bmod 2^m$. However, $-u = (2^m - u) \bmod 2^m$, which completes the validity proof for Algorithm 3b.

III. Proposed Simplified Method and its Evaluation

A. Arazi and Qi's Method

1. Exponent of 2 is a Power of 2

The main focus of this thesis will be on a procedure described by Arazi and Qi in [10] which is quite different from the methods described earlier. It is a more efficient recursive method, tailored specifically to find the inverse modulo power of 2. The method first considers the exponent of 2 to be an even number and a power of 2 although the authors have given a way to determine the inverse for other exponents as well. However, for the sake of simplicity of explanation, the exponent of 2 is considered power of 2. Let us consider the case where we need to find the value of x, where $x = y^{-1} \bmod 2^{2i}$. It can be seen that x and y are both $2i$ bit numbers while the values of y and $2i$ are known. In case the binary representation of y has more than $2i$ bits, only the lower $2i$ bits need to be considered. Let x_H and x_L denote the upper and lower i bits of x and y_H and y_L denote the upper and lower i bits of the number y respectively. $x_H x_L$ denotes the concatenation of the two i bit strings and hence the number x. Similarly $y_H y_L$ denotes the number y. Let $r = y_L^{-1} \bmod 2^i$. Arazi and Qi's method shows that if we know the value of r, the value of x can be calculated by individually calculating the upper and lower i bits of x as:

$$x_L = r \quad \text{..... (5)}$$

$$x_H = -[[(r \cdot y_L)_H + (r \cdot y_H)_L] \cdot r] \bmod 2^i, \quad \text{..... (6)}$$

where $(r \cdot y_L)_H$ and $(r \cdot y_H)_L$ denote the upper and lower half of the depicted product respectively. The final result can be obtained by concatenating the two halves as $x_H x_L$.

a. Validity

The validity of this method has been proven in [10], which for the sake the sake of completeness is also given here. Since $x = y^{-1} \bmod 2^{2i}$, we have, $x \cdot y = 1 \bmod 2^{2i}$, that is, the lower $2i$ bits of the product $x \cdot y$ (which is a $4i$ bit number) is of the form $000\ldots01$. Writing x and y in terms of their upper and lower halves, we have,

$$(x_H x_L) \cdot (y_H y_L) = x_H \cdot y_H \cdot 2^{2i} + (x_H \cdot y_L + x_L \cdot y_H) \cdot 2^i + x_L \cdot y_L$$

The lower $2i$ bits of this product is given by

$$(x_H \cdot y_L + x_L \cdot y_H) \cdot 2^i + x_L \cdot y_L$$

Since we have $x \cdot y = 1 \bmod 2^{2i}$,

$$[(x_H \cdot y_L + x_L \cdot y_H) \cdot 2^i + x_L \cdot y_L] = 1 \bmod 2^{2i} \quad \ldots\ldots (7)$$

b. Calculating x_L:

Since x_L is the lower i bits of (1) given above which should be of the form $000\ldots01$, we have,

$$x_L \cdot y_L = 1 \bmod 2^i \quad \ldots\ldots (8)$$

Thus $x_L = y_L^{-1} \bmod 2^i = r$, which, from our assumption, is already known.

c. Calculating x_H:

Since x_H is the upper i bits of (1) which should be 0, we have

$$[(x_H \cdot y_L + x_L \cdot y_H)_L + (x_L \cdot y_L)_H] \bmod 2^i = 0$$

$$(x_H \cdot x_L)_L = -[(x_L \cdot y_L)_H + (x_L \cdot y_H)_L] \bmod 2^i \quad \ldots\ldots (9)$$

Since the lower i bits k_L of any number k is also $k \bmod 2^i$ we have,

$$x_H = -[[(x_L \cdot y_L)_H + (x_L \cdot y_H)_L] \cdot y_L^{-1}] \bmod 2^i$$

Thus,

$$x_H = -[[(r \cdot y_L)_H + (r \cdot y_H)_L] \cdot r] \bmod 2^i \quad \text{..... (10)}$$

From this, the value of x can be calculated as $x = x_H x_L$. Thus if the value of $y_L^{-1} \bmod 2^i$ is known, it can be seen that the value of x can be calculated. Since we had considered n being a power of 2, to calculate the value of $y_L^{-1} \bmod 2^i$, we can consider y_L to have two halves of i bits, and this same process can be repeated recursively in that manner until y_L becomes the LSB of y, which is always 1 since y has to be odd to have an inverse modulo 2^m.

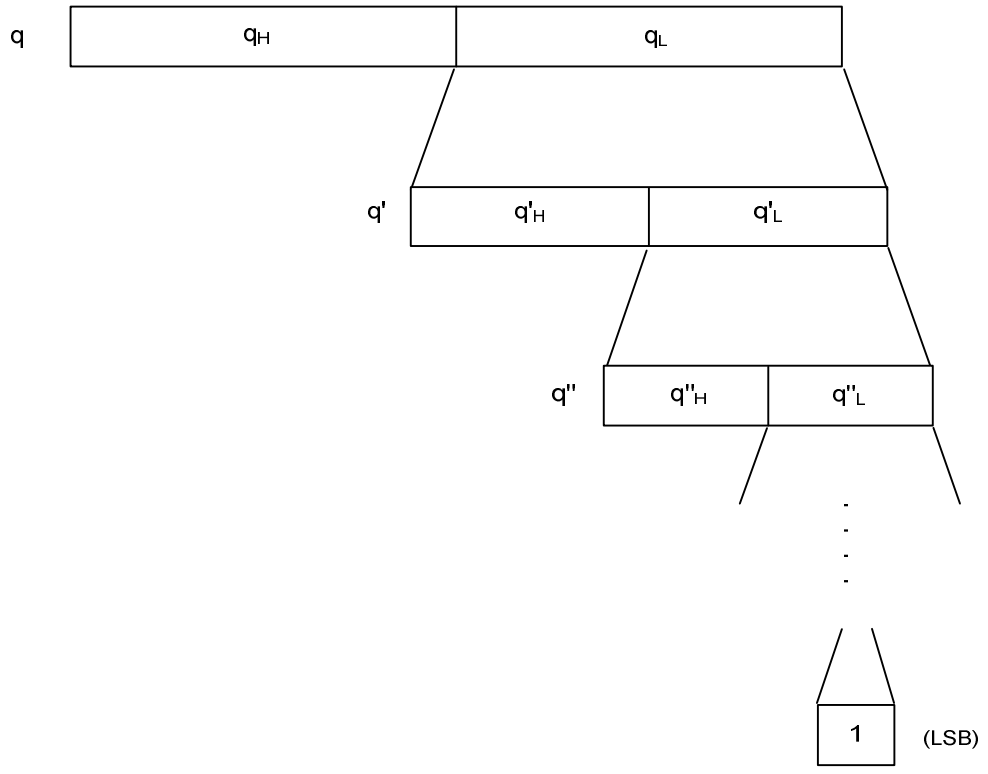


Fig. 3.1 Breakdown of the operand q

Fig. 3.1 shows how the operand q is broken down into halves recursively. The value of the inverse of q at each level of recursion can be calculated from the value of inverse of q at the next level of recursion. Using this calculated value and the value of q at the current level of recursion, its inverse is obtained using the formulas given in equations (7) and (10).

Example 1:

This result can be illustrated with a numerical example using smaller size numbers. For simplicity, the numbers have been represented in hexadecimal. However, the number system used to represent these numbers is immaterial as long as we can keep track of the binary digit representation of the numbers involved. The aim here is to find the value of $p = q^{-1} \bmod 2^{32}$, given that $q = 99F8A5EF$ and $q_L = b$, where $r = b^{-1} = (A5EF)^{-1} = 290F \bmod 2^{16}$. Using the method described above, we know that the lower half of the result, $p_L = r = 290F$. In order to calculate p_H, we need to calculate the following values:

$$(r \cdot b)_H = (290F \cdot A5EF)_H = 1A9D$$

$$(r \cdot q_H)_L = (290F \cdot 99F8)_L = BD88$$

$$\begin{aligned} p_H &= -[((r \cdot b)_H + (q_H \cdot r)_L) \cdot r] \bmod 2^{16} \\ &= -((1A9D + BD88) \cdot 290F) \bmod 2^{16} \\ &= 68D5 \end{aligned}$$

The final result p can now be obtained through concatenation. Thus $p = p_H p_L = 68D5290F$. We can see that if we multiply this number p by the original number q, we get $p \cdot q = 68D5290F \cdot 99F8A5EF = 3F0D37FD00000001$ which is $1 \bmod 2^{32}$. Hence, $p = q^{-1} \bmod 2^{32}$.

Example 2:

Another example showing the recursive way in which this algorithm can be implemented is given below. For simplicity a smaller modulus is used here. Let $q = C1D9$ be the number whose inverse is to be found with the value of the modulus as 2^{16}. Firstly the number q is divided into upper and lower halves with $q_H = C1$ and $q_L = D9$. Thus first we need the value of $q_L^{-1} \bmod 2^8$. Let this value be p' and let q_L be divided into upper and lowers halves as $q'_H = D$ and $q'_L = 9$ respectively. To find p' we need the value of $q_L'^{-1} \bmod 2^4$. Let this value be p''. Again, to calculate the value of p'' we need to divide q'_L into upper and lower halves as $q''_H = 10_2$ and $q''_L = 01_2$, which are now in binary. Again, to calculate this, we need the value of $q_L''^{-1} \bmod 2^2$. Let this value be p'''. For this, we have to divide q''_L into upper and lower halves as $q'''_H = 0$ and $q'''_L = 1$. Thus we have $q_L'''^{-1} \bmod 2 = 1$ which is the value of p'''_L, the lower half of p'''. The upper half of p''' can be calculated using equation (2) as $p'''_H = -[[(1 \cdot 1)_H + (1 \cdot 0)_L] \cdot 1] \bmod 2^2 = 0$. Hence the value of $p''' = p'''_H \mid p'''_L = 1$. Using same method, we obtain $p'' = 9$, $p' = 69_{16}$ and finally the required inverse $p = AE69$. As a verification, we can see that $p * q = 1 \bmod 2^{16}$.

2. Exponent of 2 is not a Power of 2

The method described in the previous section has been made possible because of our assumption that the exponent of the modulus is a power of 2. However, even in cases where this value is not a power of 2, the inverse can be calculated using a few extra computations [10]. Let us suppose we have $a^{-1} = b \bmod p$ with q a divisor of p. This also means that $a = b^{-1} \bmod p$. From number theory, we also have the fact that $[a \bmod q]^{-1} = b \bmod q$ which also implies $a \bmod q = b^{-1} \bmod q$. Let us suppose we need to find the value of $r = b^{-1} \bmod 2^m$ using Arazi and Qi's method. For the method to be applicable m must be a power of 2. In case m is not a power of 2, the binary representation of b is padded with

leading 0's so that the number of bits in b becomes $n = 2^{ceil(\log_2 m)}$ where $ceil(x)$ is the integer which is just greater than or equal to x. We can clearly see that the value of n is 2 raised to the power number of bits in the binary representation of m. Now the value of $w = b^{-1} \bmod 2^n$ is calculated using Arazi and Qi's method. Here it should be noted that $2^n > 2^m$ is always true since $n > m$. Hence we can say $2^m \mid 2^n$. Since we now know the value of $b^{-1} \bmod 2^n$ and 2^m divides 2^n, as per the discussion in this section we can say that $b^{-1} \bmod 2^m = w \bmod 2^m$ which is our required value. In non mathematical terms, what this means is that, if we need to find a value $a = b^{-1} \bmod 2^m$ where m is not an exact power of 2, we first find the value $b^{-1} \bmod 2^n$ where n is a power of 2 just greater than m. If we take this result modulo 2^m we get our desired result.

B. Numbers with Special Structure

When calculating the Montgomery multiplication of two numbers represented using a radix r and modulus M, it is necessary for a value M' to be precomputed. It is required for use in a step to calculate the Montgomery reduction of the intermediate values obtained during the process. This value M' is given by the operation $M' = -M \bmod r$. Since M is represented in radix r, if M_0 represents the least significant digit of M, we have $M' = -M_0 \bmod r$. This precomputed value is then multiplied to the intermediate result as a part of the operation to find the Montgomery reduction of the intermediate result. It has been shown in [22] that if we can fix the value of M' to 1 for all cases, the need for the precomputation as well as multiplication in the intermediate steps can be eliminated, thus resulting in less complexity in implementation as well as faster speed of execution. To achieve this, certain conditions have been assumed to hold true. Firstly, let the modulus M consist of n bits in its binary representation with the radix of representation being 2^w. This is usually a valid assumption in case of practical implementations.

Lemma 1: Let $M = \Delta 2^w + 1$ be an n digit positive integer in radix 2 representation, that is, $2^{n-w-1} \le \Delta < 2^{n-w}$, and let $M' = -M^{-1} \bmod 2^w$, then $M' = -1$.

Proof: Since it is clear that $M \equiv 1 \bmod 2^w$, we have $M^{-1} \bmod 2^w = 1$. Thus, $M' = -1$.

Lemma 2: Let $M = \Delta 2^w - 1$ be an n digit positive integer in radix 2 representation, that is, $2^{n-w-1} < \Delta \le 2^{n-w}$, and let $M' = -M^{-1} \bmod 2^w$, then $M' = 1$.

Proof: Here, since $M \equiv -1 \bmod 2^w$ we can see that $-M^{-1} \equiv 1 \bmod 2^w$. Thus we have $M' = 1$.

Our main concern here is the structure of the numbers involved. Let the set of numbers with the structure represented in Lemma 1 be set S1 and the set of numbers with the structure represented in Lemma 2 be set S2. In any of these cases, we can see that since Δ is an integer, $n > w$ is always true. Moreover, the lemma is true regardless of the value of w, as long as $n > w$ holds true. From the mathematical description given, we can see that the numbers in sets S1 and S2 are of the form shown below:

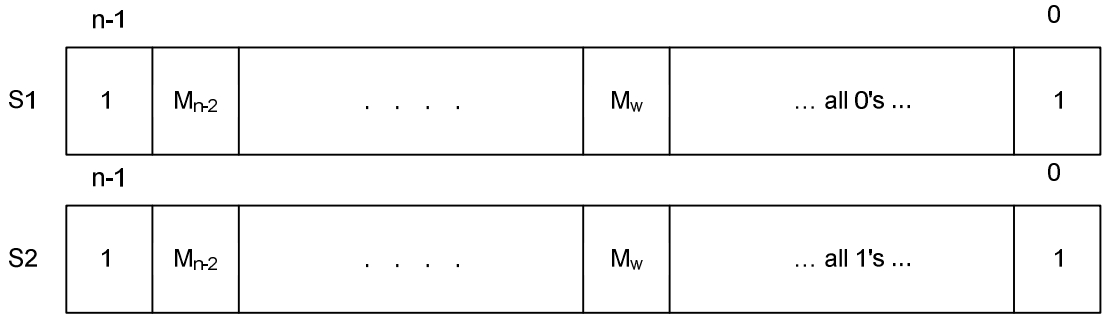


Fig. 3.2 representation of the numbers in sets S1 and S2

Fig. 3.2 shows what the numbers look like in their binary form. The numbers are n-bits wide. It can be seen that in the numbers in set S1, the most significant bit is 1, followed by a pattern of bits which represent Δ followed by $w-1$ bits of 0's with the least significant bit having a value of 1. Similarly, in the case of numbers in the set S2, the most significant bit is 1, followed by a pattern of bits representing Δ followed by w bits of 1.

C. Simplified Method for Special Numbers

This chapter presents a simplified version of the algorithm given by Arazi and Qi. It also provides an analysis of the performance of the new simplified algorithm, comparing its execution speed with the original version and other popular algorithms like Dusse and Kaliski's algorithm, straight forward method and modified extended Euclidean method. Firstly we make an assumption that $w = n/2$. As already mentioned in the previous chapter, this assumption does not invalidate the Lemmas 1 and 2. If this assumption holds true, calculation of the inverse of the n bit number M becomes easy and efficient. Let M_H and M_L denote the upper and lower $n/2$ or w bits, respectively, of the number M. From Arazi and Qi's method described in the last chapter, we can see that $M^{-1} \bmod 2^n$ can be obtained if $M_L^{-1} \bmod 2^{n/2}$ is known. Let us consider the case of M being the member of the set S1. As can be seen from Fig. 3.2, the lower w bits of a number of this structure are of the form $000\ldots01$. Thus we can see that the lower w bits of M is given by $M_L = M \bmod 2^w = (\Delta 2^w + 1) \bmod 2^w = 1 \bmod 2^w$. As the inverse of 1 in any finite field is also 1, we have, $M_L^{-1} \bmod 2^w = 1$. As the value of M_L^{-1} is known, the value of $M^{-1} \bmod 2^n$ can be calculated.

Again, let us consider the case of M being a number having the structure $\Delta 2^w - 1$, that is, a member of the set S2. In this case the lower w bits of M are given by $M_L = M \bmod 2^w = -1 \bmod 2^w$. We can see that the lower w bits of M are all 1's, that is, the number M_L is of the form $111\ldots11$. Thus M_L is a Mersenne number. A Mersenne number is a number that can be written in the form of 1 less than a power of 2. If an integer with α bits is a Mersenne number, then it has the form $2^\alpha - 1$. If we take the square of this Mersenne number modulo 2^α we get,

$$(2^\alpha - 1)^2 \bmod 2^\alpha$$

$$= (2^{2\alpha} - 2^{\alpha+1} + 1) \bmod 2^\alpha$$

$= 1 \bmod 2^{\alpha}$

The square of a Mersenne number with α bits is 1 modulo 2^{α}. Thus we can say that a Mersenne number with α bits is the inverse of itself modulo 2^{α}. Since M_L is a Mersenne number with w bits for $M \in$ S2, it is the inverse of itself modulo 2^w .and as $M_L^{-1} \bmod 2^w$ is known, the value of $M^{-1} \bmod 2^n$ can be calculated using the previously described process.

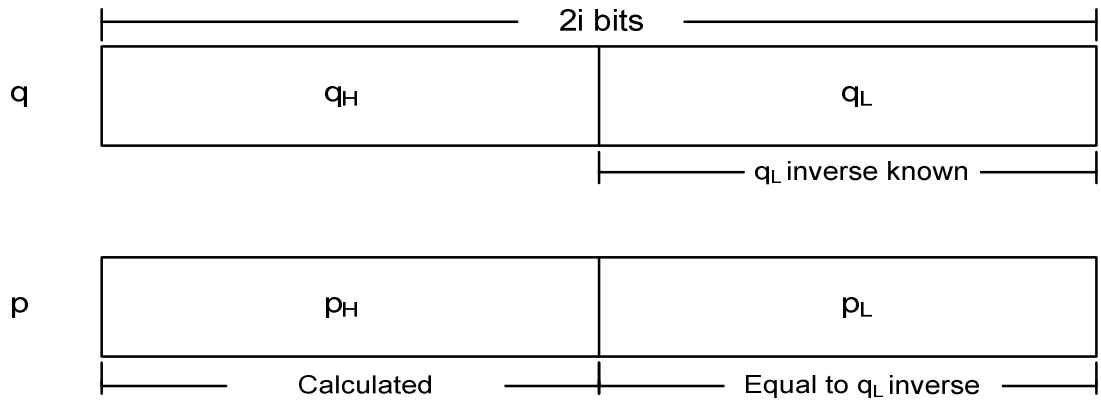


Fig. 3.3 Recursion unnecessary when value of q_L^{-1} is known

Fig. 3.3 shows how easy it becomes to calculate the value of p when we consider the numbers having the special structure as inputs. Compared to Fig. 3.1, we can see here that we do not need to break down the lower half of q, q_L, in successive recursive steps, since the value of $q_L^{-1} \bmod 2^i$ is known immediately from the beginning. In fact the most computationally intensive part of the entire operation becomes the calculation of p_H which only needs to be done once.

Example 3:

A numerical example showing the use of this method for numbers having the special structure is given here. Let $q = A45C13DE00000001$ be a 64 bit number shown here in hexadecimal notation for clarity of presentation. From its structure we can see that this number belongs to the set S1. We have $q_L = 00000001$ and $q_H = A45C13DE$ where q_L and q_H have their usual meanings. We need to find $p = q^{-1} \bmod 2^{64}$. From the simplified method, we know that $p_L = q_L^{-1} = 1$ and from equation (2) we can calculate $p_H = 5BA3EC22$. Thus we have $p = q^{-1} \bmod 2^{64} = 5BA3EC2200000001$. Again, let us consider a number in the set S2. Let $q = A45C13DEFFFFFFFF$, so that we have $q_L = FFFFFFFF$ and $q_H = A45C13DE$. Here as well, we can see that $p_L = q_L^{-1} = FFFFFFFF$ and from equation (2) we can calculate $p_H = 5BA3EC20$. Thus the inverse is $p = 5BA3EC20FFFFFFFF$.

D. Performance Evaluation

The unmodified version of this algorithm consists of three multiplication operations and are considered the most time consuming parts of this operation; the determination of the values of the expressions $(r \cdot y_L)_H$, $(r \cdot y_H)_L$ and $[[(r \cdot y_L)_H + (r \cdot y_H)_L] \cdot r] \bmod 2^i$. Each value is i bits long. The first operation requires the multiplication of two i bit values. The latter two however, each require half as much computation since they only need the lower i bits of the $2i$ bit product. Hence the execution of the process requires two multiplications of i bit values. If we consider the n bit number who's inverse modulo 2^n is to be found to be composed of k bit words, then the total number of words is given by $m = n/k$. Since k is taken to be the size of the processor word which is usually a power of 2 and we have already considered n being a power of 2, the value m in this case is an integer. If we were to use the method by halving the number of words in the lower half of y until we reach the least significant bit, at any given point of the iteration we would need to do two multiplications of

2^i bit words. Since multiplying two numbers each x words in length requires x^2 single word multiplications, multiplying 2^i bit words require $2 \cdot 2^{2i}$ single word multiplications. Since the value of i changes from 0 to $\log_2 m$ the total number of single word multiplications required is given by $2 \cdot \sum_{0}^{\log m - 1} 2^{2i}$ which simplifies to $2 \cdot \frac{(m^2 - 1)}{3}$ where m^2 gives the number of single word multiplications executed when multiplying two n bit operands (since the n bits consist of m words). Hence the computation involved in this process is two-thirds of one multiplication of n bit values. Besides this, as we recursively execute this algorithm, we will reach a point where we need to find the inverse of the least significant word. After this we need to go into bit level operations from word level operations, dividing the word into half words and so forth until we reach the least significant bit. Considering each bit level operation as a word level operation, which would be a logical choice as in the case of a software implementation, this collection of bit level operations would take a total of $2 \cdot \log k$ single word operation in addition to the amount of time required for the execution of the other word level operations. However, the total number of these bit level operations and the time required for their execution are insignificant in comparison to the other word level operations. Hence the amount of time required for execution can be approximated to $2 \cdot \frac{(m^2 - 1)}{3}$ single word multiplications. However if we consider the special condition described in this paper, we can see that no matter how many words the number y needs to be divided into, the value of $y_L^{-1} \bmod 2^i$ is always known at the first iteration itself, where y_L^{-1} is y_L itself and $i = w$. Hence for values of y lying in one of the special sets of numbers described, the method described in section 2 requires only two multiplications of w bit values. The most time consuming operation in the process has thus been reduced to two w word multiplications. Furthermore, the need to reiterate the process recursively $\log_2 n$ number of times is also eliminated. Further reducing the cost, for example, of subroutine calls in case of a software implementation or the total execution time required in case of hardware implementations.

The algorithms from Algorithm 1 through 3 were implemented in C, using libtommath large number library to compare the efficiency of the various algorithms, Arazi and Qi's original algorithm and the modified algorithm. It was seen that the modified algorithm was faster than the other algorithms for most of the cases. Another tool was also developed to generate random numbers of a given bit length, to use for testing the algorithms. Although for smaller sized numbers such as 32 bit and 64 bit numbers, Algorithm 3 was seen to be faster than the modified algorithm in some cases, as the number of bits in the input was increased it was seen that the modified algorithm was definitely superior in terms of execution speed. The programs were written in C and compiled using gcc version 4.4.1 for both Linux and Windows Operating System environments, on an Intel Core i5 – 2500K computer with 3.30 GHz processor frequency. Data showing how many times the modified algorithm is faster (or in very few cases, slower) than the other algorithms considered, was recorded and tabulated here. It is also shown in graphical form as follows.

Algorithm	32 bits	64 bits	128 bits	256 bits	512 bits	1024 bits
Arazi and Qi's method	5.34	6.2	6.91	7.27	6.19	5.04
Dusse and Kaliski's method	1.21	3.36	8.37	22.91	64.26	129.33
Straightforward method	1.57	4.39	10.02	22.56	44.61	53.29
Modified extended Euclidean method	0.008	0.51	1.55	4.25	8.51	10.22

Table 3.1 Execution speed comparison for set 1 numbers in windows environment

Algorithm	32 bits	64 bits	128 bits	256 bits	512 bits	1024 bits
Arazi and Qi's method	3.66	4.18	4.58	4.27	2.92	1.99
Dusse and Kaliski's method	0.38	1.48	4.14	11.15	26.89	54.84
Straightforward method	0.65	2.1	5.06	10.78	18.06	20.88
Modified extended Euclidean method	-0.46	-0.36	-0.05	0.6	1.17	1.35

Table 3.2 Execution speed comparison for set 2 numbers in windows environment

Algorithm	32 bits	64 bits	128 bits	256 bits	512 bits	1024 bits
Arazi and Qi’s method	5.06	5.88	6.50	6.88	6.2	5.94
Dusse and Kaliski's method	1.20	3.15	7.89	23.09	66.58	157.12
Straightforward method	1.64	4.27	9.50	20.07	37.85	47.33
Modified extended Euclidean method	-0.03	0.34	1.23	3.13	7.02	8.54

Table 3.3 Execution speed comparison for set 1 numbers in Linux environment

Algorithm	32 bits	64 bits	128 bits	256 bits	512 bits	1024 bits
Arazi and Qi’s method	3.45	3.99	4.5	4.5	3.41	2.7
Dusse and Kaliski's method	0.26	1.32	3.88	11.315	31.57	71.06
Straightforward method	0.5	1.91	4.62	9.21	15.16	18.87
Modified extended Euclidean method	-0.52	-0.45	-0.17	0.31	0.89	1.17

Table 3.4 Execution speed comparison for set 2 numbers in Linux environment

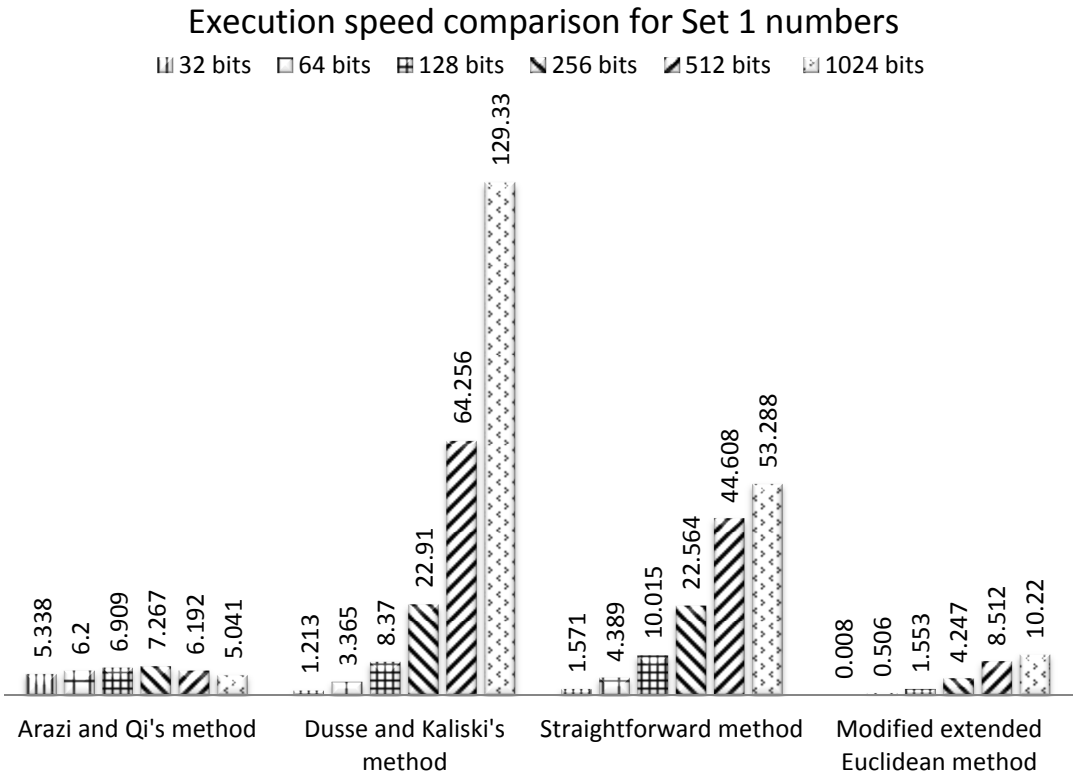


Fig. 3.4 Performance comparison for execution speed of algorithms using input in set S1 in the windows environment

Fig. 3.4 compares Arazi and Qi's original method, Dusse and Kaliski's algorithm, the straight forward method and the modified extended Euclidean method with the proposed simplified method. The input numbers are of the form that are from set S1. It can be seen that as the size of the input number increased, the performance of the simplified method with respect to the other methods also increased. It can be seen that the maximum performance improvement was seen for 1024 bit large inputs over the Dusse and Kaliski's method. The simplified algorithm was around 129 times faster than Dusse and Kaliski's method for the same inputs.

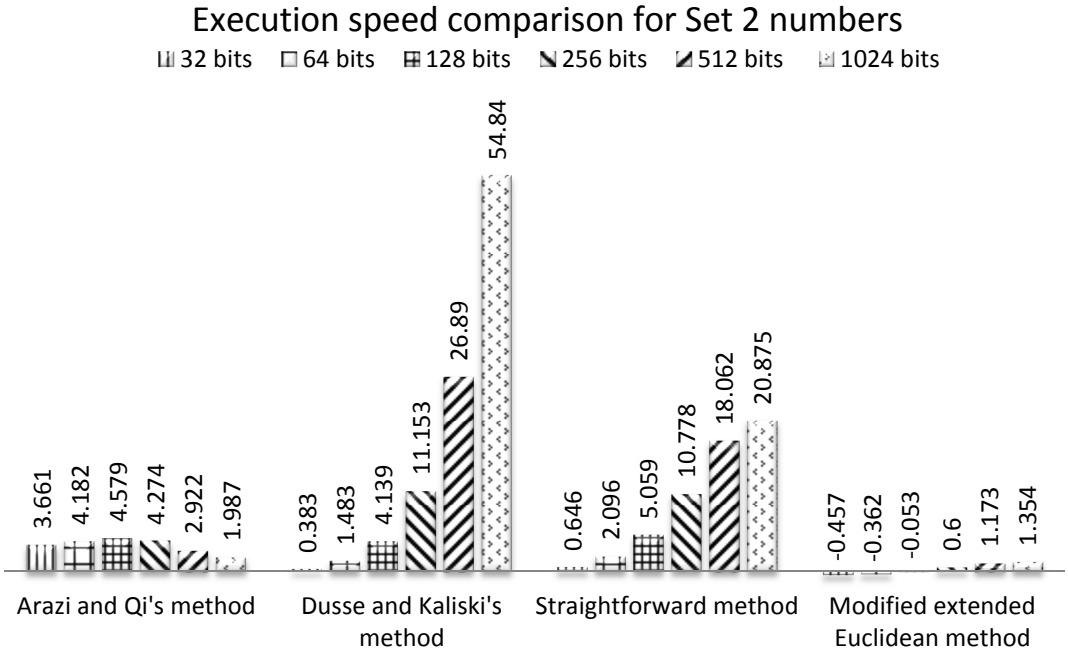


Fig. 3.5 Performance comparison for execution speed of algorithms using input in set S2 in the windows environment

Fig. 3.5 shows a comparison of performance for numbers that lie in set S2. Here also we can see that maximum performance increase is seen over Dusse and Kaliski's method. It can be seen that, although for smaller sized numbers, modified extended Euclidean algorithm executed faster than the simplified algorithm described here, as the size of the input number became larger, the improvement supplied by the modified algorithm became more apparent.

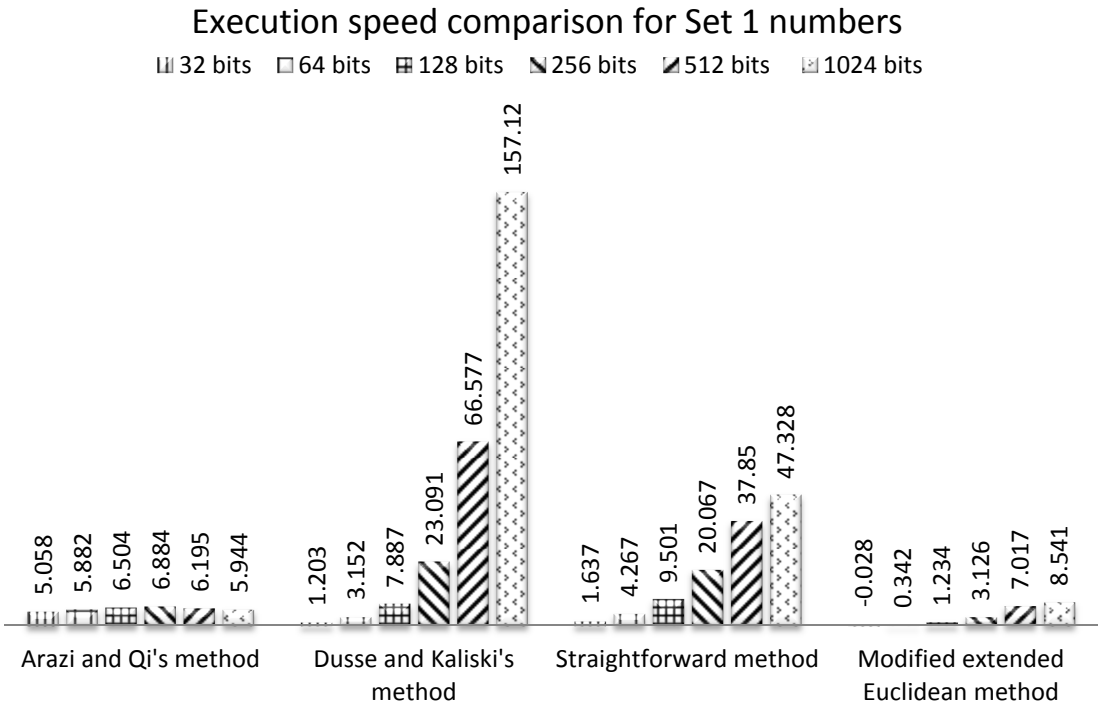


Fig. 3.6 Performance comparison for execution speed of algorithms using input in set S1 in the Linux environment

Fig. 3.6 compares the speed of execution of the four algorithms with the simplified method for numbers in the set S1, this time in a Linux environment. It can be seen that the overall trend of the data is not much different in this case either, with the most improvement being seen over Dusse and Kaliski's method for 1024 bit numbers.

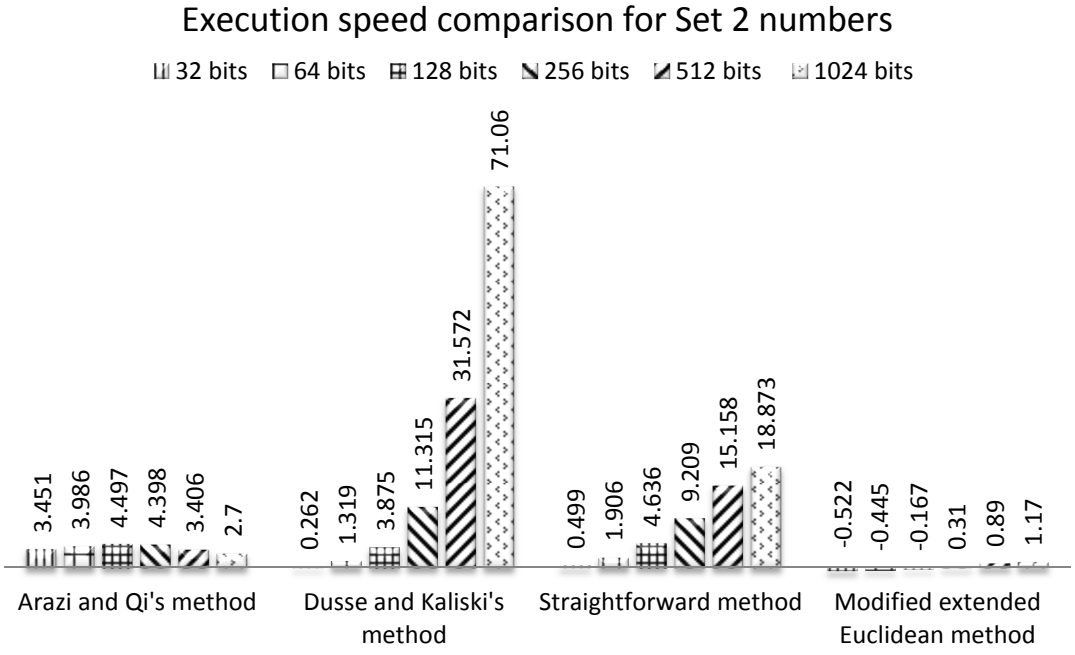


Fig. 3.7 Performance comparison for execution speed of algorithms using input in set S2 in the Linux environment

Fig. 3.7 shows the comparison of performance of the four algorithms with the simplified method for numbers in the set S2 in a Linux environment. As expected the overall trend of data was not much different, with the most improvement being seen over Dusse and Kaliski's method for 1024 bit numbers. Here as well, although for smaller size inputs the modified extended Euclidean algorithm was seen to be faster, as the size of input increased an improvement in performance of the simplified algorithm was also seen as expected.

IV. Conclusion

Most implementations of Public Key Cryptosystem (PKCS) utilize different types of modular operations. Modular inverse is one such operation and is a time consuming operations compared to addition and subtraction, especially when the size of input numbers is large. A special case is inversion modulo a power of 2. Dusse and Kaliski's method, the straight forward method and the extended Euclidean method are some of the popular established ways of finding such an inverse. Arazi and Qi had proposed a recursive algorithm which is more efficient than these established methods.

This thesis proposes a simplified version of Arazi and Qi's method, by removing the recursion process. For the same inputs, the simplified method is able to execute faster than the previously mentioned methods, especially when the size of the input is large. Arazi and Qi's method divides the input numbers into halves recursively up to the MSB. For the method proposed in this thesis, if the input numbers have a certain structure in their binary representation, then the recursion can be eliminated. This increases the speed of execution of the simplified algorithm.

The special structures are of two types. Depending on them, an input number belongs to either the set S1 or S2. The experimental results show that the inverse can be found faster than other algorithms if the proposed simplified method is used if the input is from one of these sets. This increase in execution speed is more significant when the size of the input is large. We can see from the experimental data that for inputs of smaller size such as 32 or 64 bits, the speed gain was not much significant. For example, in case of numbers belonging to set S2, the negative value indicates that the extended Euclidean algorithm may be faster than the proposed modified method. However, the speed gain becomes much significant as the size of the input increases. For instance, for inputs of size 1024 bits belonging to set S1, it can be seen that the simplified method is 5 times faster than Arazi and Qi's method, 129.3 times faster than Dusse and Kaliski's algorithm, 53.3 times faster than the straight forward method and 10.2 times faster than the modified extended Euclidean algorithm.

It should be noted however that this performance enhancement comes at a price. The trade off in this case is that the simplified algorithm is only applicable when the input numbers have a special structure in their binary representation and fall in either set S1 or S2. If the input does not fall in either of these sets then the simplified method cannot be applied to obtain the execution speed increase.

As a future enhancement, work may be done in designing a hardware circuit for this method. The paper by Arazi and Qi mostly talks about software implementation and has considered software based implementation as a basis of performance comparison. Hence one possible way in which this work may be extended is by developing a hardware realization of Arazi and Qi's recursive method.

References

[1] A. V. Oppenheim, A. S. Willsky and S. H. Nawab, *Signals and Systems*, 2nd ed., Prentice Hall, 1997.

[2] H. P. van Tillsborg, *Encyclopedia of Cryptography*. Springer Science+Business Media Inc, 2005.

[3] C. Kaufman, R. Perlman and M. Speciner, *Network Security Private Communication in a Public World*. 2nd edition, Prentice Hall, 2002.

[4] A. J. Menenzes, P.C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, Inc., 1997.

[5] D. E. Knuth, *The Art of Computer Programming Seminumerical Algorithms*. 2nd ed., vol. 2, Addison-Wesley Publishing Company, 1984

[6] B. L. Kaliski, "The Montgomery inverse and It's Applications," *IEEE Trans. Comput.*, vol. 44, no. 8, pp 1064-1065, Aug. 1995.

[7] E. Savas and C. K. Koc, "The Montgomery Modular Inverse—Revisited," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 763-766, Jul. 2000.

[8] A. A. A. Gutub, A. F. Tenka, E. Savas and C. K. Koc, "Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^n)," in *Cryptographic Hardware and Embedded Systems - CHES 2002*, B.S. Kaliski Jr. et al., Ed. 2003, Lecture Notes in Computer Science, No. 2523, pp. 484-499, Springer, Verlag Berlin Heidelberg 2003.

[9] D. Hankerson, A. Menenzes and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[10] O. Arazi and H. Qi, "On calculating multiplicative inverses modulo 2^m," *IEEE Trans. Comput.,* vol. 57, no. 10, pp. 1435-1438, Oct. 2008

[11] S. R. Dusse and B. S. Kaliski, "A Cryptographic Library for the Motorola DSP5600," Advances in Cryptology, *Proc. Ann. EuroCrypt Conf.*, pp. 230-244, 1990.

[12] W. Stallings, *Cryptography and Network Security Principles and Practices.* 3rd ed., Pearson Education Inc, 2003.

[13] C. Parr and J. Pelzl, *Understanding Cryptography A Textbook for Students and Practitioners*. Springer-Verlag, 2010.

[14] W. Diffie and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 644-656, 1967.

[15] D. R. Stinson, *Cryptography Theory and Practice.* 2nd ed., Chapman and Hall/CRC, 2002.

[16] K. Bentahar and N. P. Smart, "Efficient 15,360-bit RSA Using Woop-Optimised Montgomery Arithmetic," *Cryptography and Coding 2007*, LNCS 4887, pp 346-363, 2007

[17] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba Algorithm for Efficient Implementations," Technical Report, University of Ruhr, Bochum, Germany, 2003. Available: http://eprint.iacr.org/2006/224

[18] M. E. Kaihara and N. Takagi, "Bipartite modular multiplication," *IEEE Trans. Comput.*, vol. 57, no. 2, pp. 157-164, Feb. 2008.

[19] C. K. Koc, "High-Speed RSA Implementation," RSA Laboratories, Tech. rep. TR-201, Nov. 1994.

[20] P. L. Montgomery, "Modular Multiplication Without Trial Division," *Mathematics of Computation,* vol. 44, no. 170, pp. 519-521, 1985.

[21] C. K. Koc, "Montgomery reduction with Even Modulus," *IEEE Proc. Comput. Digit. Tech.*, vol. 141, no. 5, Sept. 1994.

[22] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster interleaved modular multiplication based on Barrett and Montgomery reduction methods," *IEEE Trans. Comput.*, vol. 59, no. 12, pp. 1715-1721, Dec. 2010.

Acknowledgement

I would like to express my deep and sincere gratitude to my academic supervisor, Prof. Kim, Young Sik. His invaluable support, encouragement, supervision, personal guidance, and useful suggestions throughout the course of my research have provided a good basis for the completion of this thesis.

I owe my most sincere gratitude to Prof. Han, Seung Jo for his precious support, encouragement and cooperation.

I wish to express my warm and sincere thanks to the thesis committee members, Prof. Park , Jong An and Prof. Pyun, Jae Young for their detailed and constructive comments, and for their important support throughout this work.

I also like to express thanks to all my lab mates of Computer Network and Information Security Lab, Information Theory and Security Lab and all friends in Korea for their support and cooperation.

I owe my loving thanks to my parents. Without their encouragement and understanding it would have been impossible for me to finish this work. I owe my special gratitude to my family and friends at home for their continued support.