Abstract— Modular multiplication is the fundamental operation in most public-key cryptosystem. Therefore, the efficiency of modular multiplication directly affects the efficiency of whole crypto-system. This paper presents an efficient modular multiplication algorithm for large integer. The proposed algorithm integrates with three existing algorithm, Barrett Algorithm and Montgomery algorithm for modular multiplication, Toom-Cook algorithm for multiplication. This algorithm execution done in parallel way so that enhance the performance. These algorithms Analysis with respect to their performance and compare to other modular multiplication algorithms.

Index Terms— Barrett algorithm, Bipartite modular multiplication, Karatsuba multiplication algorithm, Montgomery algorithm, Toom-Cook multiplication, Tripartite modular multiplication.

I. INTRODUCTION

Public Key Cryptography (PKC) introduced by Diffie and Hellman [1] in the mid-1970s. Many cryptographic protocols, such as the RSA scheme [2], ElGamal [3], Diffie-Hellman key exchange, and DSA [4], are based on modular arithmetic operations.

The efficiency of a particular cryptosystem will depend on a number of factors, such as parameter size, time-memory tradeoffs, available processing power, parallel computing, software and/or hardware optimization, and mathematical algorithms. An efficient implementation of this operation is the key to high performance. A basic operation in public key cryptosystems is the modular multiplication of large numbers.

This paper deals with different modular multiplication algorithms namely Barrett algorithm [11], Montgomery algorithm [9], Bipartite algorithm [5], Tripartite algorithm [19] and proposed algorithm. Barrett and Montgomery algorithms are widely used today. Barrett algorithm output in this algorithm is (X.Y)modM and this algorithm required preprocessed value. Montgomery modular multiplication algorithm output is (X.Y)RmodM and also required preprocessed value. Bipartite modular multiplication integrates Barrett and Montgomery method these methods execution in parallel way. Tripartite modular multiplication use Karatsuba multiplication for multiplication of two large number and two efficient Barrett and Montgomery algorithms compute modular multiplication these method also execution in parallel way.

The proposed modular multiplication algorithm that efficiently integrates three existing algorithms, Barrett modular multiplication, Montgomery modular multiplication and Toom-Cook multiplication, this proposed algorithm divide into two step multiplication and modular multiplication step. Multiplication step Toom-cook algorithm is used. Modular multiplication step Barrett and Montgomery algorithms are used in parallel way. The proposed algorithm minimizes the number of single-precision multiplication enable more than three way parallel computation.

The remainder of this paper is structured as follows. Section 2 describes Barrett algorithm, Montgomery algorithm, Bipartite algorithm and Tripartite algorithm. In Section 3, our proposed algorithm is introduced. Software implementation results are introduced in Section 4. Section 5 concludes the paper.

II. RELATED WORK

These algorithms for modular multiplication are described for use with large nonnegative integers expressed in radix b notation, where b can be any integer ≥ 2. Given a modulus M and two elements X, Y ∈ ZM where ZM is the ring of integers modulo M we define the ordinary modular multiplication as XY mod M

Mathematical representation of X, Y and M is inputs of modular multiplication algorithms.

\[ M = \sum_{i=0}^{k-1} m_i b^i \quad 0 < m_i < b \text{ and } 0 \leq m_i < b, \text{ for } i=0,1,\ldots,k-1 \]

\[ X = \sum_{i=0}^{k-1} x_i b^i \quad 0 < x_i < b \text{ and } 0 \leq x_i < b, \text{ for } i=0,1,\ldots,k-1 \]

\[ Y = \sum_{i=0}^{k-1} y_i b^i \quad 0 < y_i < b \text{ and } 0 \leq y_i < b, \text{ for } i=0,1,\ldots,k-1 \]

These algorithms for performing the modular multiplication and analyze their time and space requirements. The analysis is performed by counting the total number of multiplications, additions, subtractions, and memories read and write operations in terms of the input size parameter k. They are counted to calculate the proportion of the memory access time in the total running time of the modular multiplication algorithm. In our analysis, loop establishment and index computations are not taken into account. The space analysis is performed by counting the total number of words used as the temporary space. However, the space required keeping the input and output values.

Amar Mandal, Rupali Syal
A. BARRETT MODULAR MULTIPLICATION

P. Barrett [11] introduced the idea of estimating the quotient \( S/M \) with operations that either are less expensive in time than a multiprecision division by \( M \) (viz., 2 divisions by a power of band a partial multiprecision multiplication), or can be done as a pre calculation for a given \( M \) (viz., \( U = b^k/M \), i.e., \( U \) is a scaled estimate of the modulus' reciprocal). The estimate \( q \) of \( S/M \) is obtained by replacing the floating point divisions in \( q = \left( S/b^{2k-1} \mod b^k/M \right) \) by integer divisions \( \hat{q} = \left( S/b^{2k-1} \mod b^k/M \right) \).

This estimate will never be too large and, if \( k < t \leq 2k \), the error is at most two: \( S/M - 2 \leq \hat{q} \leq S/M \), for \( k < t \leq 2k \).

The best choice for \( t \), resulting in the least single precision multiplications and the smallest maximal error, is \( k+1 \), which also was Barrett's original choice. An estimate \( \hat{r} \) for \( S \mod M \) is then given by \( r = x - qm \), or, as \( r < b^{k+1} \) (if \( b > 2 \)), by \( \hat{r} = (S \mod b^{k+1} - (qm) \mod b^{k+1}) \mod b^{k+1} \), which means that once again only a partial multiprecision multiplication is needed. At most two further subtractions of mare required to obtain the correct remainder.

BARRETT MODULAR MULTIPLICATION ALGORITHM

Input: \( X=(x[k-1],x[k-2],...,x[1],x[0])_b \), \( Y=(y[k-1],y[k-2],...,y[1],y[0])_b \), \( M=(m[k-1],m[k-2],...,m[1],m[0])_b \), \( U=(u[k-1],u[k-2],...,u[1],u[0])_b \), \( b \geq 2 \)

Output: \( XY \mod M \)

1. \( S \leftarrow XY \);
2. \( q \leftarrow ((S \mod b^{k+1})U) \mod b^{k+1} \);
3. \( S \leftarrow S \mod (QM) \mod b^{k+1} \);
4. if \( S < 0 \) then
5. \( S \leftarrow S + b^{k+1} \);
6. while \( (S \geq M) \) do
7. \( S \leftarrow S - M \);

This Algorithm requires \( 3k^2 \) multiplications, \( 6k^2+k+1 \) additions, \( 9k^2+2k+2 \) reads, \( 3k^2+4k \) writes and uses \( 2k+1 \)words of memory space.

B. MONTGOMERY MODULAR MULTIPLICATION

The Let \( R > M \) be an integer relatively prime to \( M \) such that computations modulo \( R \) are easy to process: \( R = b^k \). Notice that the condition \( \gcd(M, b) = 1 \) means that this method cannot be used for all moduli. In case \( b \) is a power of 2, it simply means that \( m \) should be odd. The m-residue with respect to \( R \) of an integer \( X < M \) is defined as \( XR \mod M \). The set \( \{ XR \mod M | 0 \leq x < M \} \) clearly forms a complete residue system. The Montgomery reduction of \( X \) is defined as \( XR^{-1} \mod M \), where \( R^{-1} \) is the inverse of \( R \) modulo \( M \), and it is the inverse operation of the m-residue transformation. It can be shown that the multiplication of two m-residues followed by Montgomery reduction is isomorphic to the ordinary modular multiplication. The rationale behind the m-residue transformation is the ability to perform a Montgomery reduction \( (XR^{-1}) \mod M \) for \( 0 \leq X < RM \) in almost the same time as a multiplication. In this algorithm required one pre compute value \( M' = M^{-1} \).

MONTGOMERY MODULAR MULTIPLICATION ALGORITHM

Input: \( X=(x[k-1],x[k-2],...,x[1],x[0])_b \), \( Y=(y[k-1],y[k-2],...,y[1],y[0])_b \), \( M=(m[k-1],m[k-2],...,m[1],m[0])_b \), \( M'=(m'[k-1],m'[k-2],...,m'[1],m'[0])_b \), \( b \geq 2 \)

Output: \( XY^{-1} \mod M \)

1. \( S \leftarrow XY \);
2. for \( (i = 0; i < k; i++) \) do
3. \( t \leftarrow (S, M')_b \mod b^i \);
4. \( S \leftarrow S + t \cdot Mb^i \);
5. \}
6. \( S \leftarrow S \div b^i \);
7. if \( (S \geq M) \) then
8. \( S \leftarrow S - M \);

This algorithm (with the slight improvement above) requires \( 2k^2+k \) multiplications, \( 4k^2+4k+2 \) additions, \( 6k^2+7k+2 \) reads, and \( 2k^2+5k+1 \) writes, including the final multi-precision subtraction, and uses \( k + 3 \) words of memory space. The Montgomery representation of an integer \( X \), denoted by \( X_{Mont} \), can be computed by performing a Montgomery multiplication on \( X \) and \( R' \), denoted by \( Mont(X,R') \), resulting in \( X_{Mont} = Mont(X,R') = (XR^{-1} \mod M) \mod M = (X \mod M) \mod M' \). After computing the Montgomery multiplication of two operands in Montgomery representation, the result is also in Montgomery representation and can be converted back by multiplication with \( R'^{-1} \), which comes down to Montgomery multiplication with 1.Computation of the result: \( T = Mont(X_{Mont}, Y) = (X \cdot Y \cdot R^{-1}) \mod M = (X \cdot Y) \mod M' \).

This means that two Montgomery multiplications are needed for one modular multiplication. That is why the use of Montgomery multiplication is only interesting when many consecutive modular multiplications need to be performed.

C. BIPARTITE MODULAR MULTIPLICATION

In Bipartite Modular multiplication both Barrett and Montgomery algorithms are used in this algorithm X is dividing in to two parts upper parts calculate using Montgomery algorithm and lower part calculate using Barrett algorithm .The bipartite algorithm was introduced for the purpose of a two-way parallel computation [6]. It uses two custom modular multipliers, a Barrett modular multiplier and a Montgomery multiplier, in order to improve the speed. By combining a Barrett modular multiplication with Montgomery modular multiplication, it splits the operand multiplier into two parts and processes them in parallel, increasing the calculation speed. Parallel execution of this method with the help of fork() system call in Linux operating system. The calculation is performed using Montgomery residues defined by a modulus M and a Montgomery radix R.
R < M. Next, we outline the main idea of the bipartite algorithm. Let R = b^i for some 0 < i < k. Consider the multiplier Y to be split into two parts Y_1 and Y_0 so that Y = Y_1R + Y_0. Then, the Montgomery multiplication modulo M of the integers X and Y can be computed as follows:

\[ XY^1 \mod M = (XY + Y_0R^1) \mod M \]

\[ =((XY \mod M) + (XY \mod M)) \mod M \]

\[ = Barrett(M(X,Y) + MontM(X,Y)) \]

Let l = (k/2) Then this algorithm with the slight improvement above) requires (5/2^e) multiplications, (5^k/5/2^e+1) additions, (15/2^e+19/2^e+5) reads, and (5/2^e+17/2^e+3) writes 5k±3 subtraction, and uses 2k + 1 words of memory space. In this algorithm use both Montgomery and Barrett methods execution in parallel way so that enhance the speed.

D. TRIPARTITE MODULAR MULTIPLICATION

Tripartite modular multiplication algorithm divides into two step multiplication and modular multiplication, first multiplication step using Karatsuba algorithm and split in three parts and second part modular multiplication part execution of these three parts by using Barrett and Montgomery modular multiplication in parallel way. The first multiplication step computes by Karatsuba algorithm. The Karatsuba algorithm is a fast multiplication algorithm. It reduces the multiplication of two k-digit numbers to at most 3k log_2 3 \approx 3k^{1.585} single-digit multiplications in general (and exactly k log_2 3 when k is a power of 2). It is therefore faster than the classical algorithm, which requires k^2 single-digit products. The basic step of Karatsuba's algorithm is a formula that allows us to compute the product of two large numbers X and Y using three multiplications of smaller numbers, each with about half as many digits as X or Y, plus some additions and digit shifts. Let X and Y are represented as n-digit strings in some base B. For any positive integer l less than k, one can split the two given numbers as follows

\[ R = B^l \]

\[ X = X_0 + X_1 \]

\[ Y = Y_0 + Y_1 \]

Where \( X_0 \) and \( Y_0 \) are less than R. The product is then

\[ XY = (X_0 + X_1)(Y_0 + Y_1) \]

\[ = Z_0R^1 + Z_1(R + Z_0) \]

Where

\[ Z_0 = X_0Y_0 \]

\[ Z_1 = X_1Y_0 + X_0Y_1 = (X_1 + X_0)(Y_1 + Y_0) + Z_0 \]

Karatsuba observed that XY can be computed in only three multiplications, and few extra additions:

First step multiplication of number and splitting three parts \( Z_0, Z_1, Z_2 \) Modular multiplication step compute this three parts as follows:

\[ (XY)^1 \mod M = ((Z_0R^1 + Z_1R + Z_0) \mod M \]

\[ =((Z_0R^1 + Z_1R + Z_0) \mod M + Z_0R^1) \mod M \]

\[ = Barrett(M(Z_0, R) + Barrett(M(Z_1, 1) \mod M) \]

To obtain a high-speed implementation, one can compute these three different terms in parallel. We take l=k/2 for calculation. Modular multiplication step two Barrett methods and one Montgomery requires. this step requires 11/4k^2 + k multiplications, 11/2k^2+21/2k+8 additions, 31/4k^2+16k+10 reads, and 3/2k^2+25/2k+6 writes 7k+5 subtraction and first multiplication step required 3/4k^2 multiplications, k^2+2k additions, 9/4k^2+4k reads, and 3/2k^2+7/2k-4 writes k subtraction. The all above algorithms they are slightly more number of operations in read, write, multiplication, subtraction and addition but this algorithm computes parallel way first and second parts using Barrett algorithm third term using Montgomery algorithm, modular multiplication step of This algorithm execution in parallel way so that time consuming is less than other algorithm.

III. THE PROPOSED MODULAR MULTIPLICATION

The proposed modular multiplication algorithm divides into two step multiplication and modular multiplication. Multiplication step using Toom-Cook algorithm and split in five parts and in modular multiplication step compute of these five parts by using Barrett and Montgomery modular multiplication in parallel way.

Multiplication step computes by Toom-Cook multiplication algorithm. Given two large integers, X and Y, Toom-Cook splits up X and Y into t smaller parts each of length l, and performs operations on the parts. Toom-3 is only a single instance of the Toom–Cook algorithm, where \( t = 3 \). Therefore, Toom-3 reduces 9 multiplications to 5, and runs in \( \Theta(n^{5/3} \log(3)) \), about \( \Theta(n^{1.67}) \). In general, Toom-r runs in \( \Theta(c(t) n^r) \), where \( c = \log(2r - 1) / \log(t) \) is the time spent on sub-multiplications, and \( c \) is the time spent on additions and multiplication by small constants. The Karatsuba algorithm is a special case of Toom–Cook, where the number is split into two smaller ones. It reduces 4 multiplications to 3 and so operates at \( \Theta(n \log(3))^3 \), which is about \( \Theta(n^{1.585}) \). Ordinary long multiplication is equivalent to Toom-1, with complexity \( \Theta(n^2) \).

In a typical large integer implementation, each integer is represented as a sequence of digits in positional notation, with the base or radix set to some (typically large) value \( b \), (in a computer implementation, \( b \) would typically be a power of 2 instead). Say the two integers being multiplied are: The base \( B = b^i \), such that the number of digits of both \( m \) and \( n \) in base \( B \) is at most \( t \), then separate \( m \) and \( n \) into their base \( B \) digits \( m_i \) and \( n_i \). Then use these digits as coefficients in degree \( t \)–1 polynomials \( p \) and \( q \), with the property that \( p(B) = m \) and \( q(B) = n \). Therefore, the product \( r(x) = p(x)q(x) \), our answer will be \( r(B) = mn\). In the case where the numbers being multiplied are of different sizes, it's useful to use different values of \( t \) for \( m \) and \( n \), which would call \( t_m \) and \( t_n \). The number of elementary operations (addition/subtraction) can be reduced. Executed here over the first operand (polynomial \( p \)) of the running example is the following:

\[ p_0 = m_0 + m_2 \]
\[ p(0) = m_0 \]
\[ p(1) = p_0 + m_1 \]
\[ p(-1) = p_0 + m_1 \]
\[ p(-2) = (p(-1) + m_2) \times 2 - m_0 \]
\[ p(x) = m_2 \]

This sequence requires five addition/subtraction operations, one less than the straightforward evaluation. In practical implementations, as the operands become smaller, the algorithm will switch to the Schoolbook long multiplication. Letting \( r \) be the product polynomial:

\[ r(0) = p(0)q(0) \]
\[ r(1) = p(1)q(1) \]
\[ r(-1) = p(-1)q(-1) \]
\[ r(-2) = p(-2)q(-2) \]
\[ r(x) = p(x)q(x) \]

A difficult design challenge in Toom–Cook is to find an efficient sequence of operations to compute this product; one sequence given by Bodrato[14] for Toom-3 is the following.

\[ r_0 = r(0) \]
\[ r_2 = r(x) \]
\[ r_3 = r(\tau - 2) - r(1) / 3 \]
\[ r_4 = r(1) - r(1) / 2 \]
\[ r_5 = r(-1) - r(0) \]
\[ r_7 = r_2 - r_1 / 2 + 2r(\alpha) \]
\[ r_8 = r_2 + r_1 - r(\alpha) \]
\[ r_9 = r_1 - r_3 \]

Now product polynomial \( r \),
\[ r(x) = r_0 + r_1 x^2 + r_3 x^4 + r_5 x^6 + r_7 x^8 \]

Finally, evaluate \( r(B) \) to obtain our final answer. This is straightforward since \( B \) is a power of \( b \) and so the multiplications by powers of \( B \) are all shifts by a whole number of digits in base \( b \).

First step multiplication of number and splitting parts, Modular multiplication step compute these parts as follows.

Let \( B = R \)
\[ r = r_0 + r_1 R + r_2 R^2 + r_3 R^3 + r_4 R^4 \]
\[ r(R)R^2 = (r_0 R^2 + r_1 R^3 + r_2 R^4 + r_3 R^5 + r_4 R^6)R^2 \]
\[ = r_0 R^4 + r_1 R^5 + r_2 R^6 + r_3 R^7 + r_4 R^8 \]

Modular multiplication with \( M \) in both sides as follows:

\[ (r(R)R^2) \mod M = (r_0 R^2 + r_1 R^3 + r_2 R^4 + r_3 R^5 + r_4 R^6) \mod M \]
\[ = \text{MontM}(\text{MontM}(r_0, 1), 1) + \text{MontM}(r_1, 1) + \text{BarrettM}(r_2, 1) + \text{BarrettM}(r_3, R) + \text{BarrettM}(r_4, R^2) \]

To obtain a high-speed implementation, one can compute these five different terms in parallel, where \( B = b^i \) for calculation of this algorithm we take \( i = k / 3 \). Second modular multiplication part having three Barrett algorithms and three Montgomery algorithms requires \( 29/18k^2 + 2k \) multiplications, \( 29/3k^2 + 11k = 12 \) additions, 17/6k + 34/3k + 16 reads, and 17/18k + 25/3k + 10 writes 11/3k + 8 subtraction. In the above all algorithms they are slightly difference in there number of operations in read, write, multiplication, subtraction and addition, this algorithm execution in parallel way so that time consuming is less than other algorithms.

IV. RESULT

Use After Software performance; Execution times for the modular multiplication of a 2k-digit number modulo a k-digit modulus \( M \) for the five modular multiplication algorithms compared to the execution time of a \( k \times k \)-digit multiplication \( (b = 2) \), on a 1.73 GHz Intel Celeron R based PC with gcc 4.5.2 (Ubuntu/Linux 4.5.2-8ubuntu4).

RESULT

<table>
<thead>
<tr>
<th>Length of M in bits</th>
<th>Barre t</th>
<th>Montgomery</th>
<th>Bipartite</th>
<th>Tripartite</th>
<th>Proposed Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>8</td>
<td>9</td>
<td>22</td>
<td>30</td>
<td>47</td>
</tr>
<tr>
<td>256</td>
<td>42</td>
<td>50</td>
<td>11</td>
<td>50</td>
<td>52</td>
</tr>
<tr>
<td>512</td>
<td>61</td>
<td>59</td>
<td>20</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>1024</td>
<td>94</td>
<td>105</td>
<td>89</td>
<td>60</td>
<td>74</td>
</tr>
<tr>
<td>2048</td>
<td>173</td>
<td>143</td>
<td>153</td>
<td>120</td>
<td>91</td>
</tr>
</tbody>
</table>

These observations are confirmed by a software implementation of these algorithms, see in Table. The implementation is written in ANSI C [4] and hence should be portable to any computer for which an implementation of the ANSI C standard exists. All figures in this article are obtained on a 1.73 GHz Intel Celeron R based PC using the 32-bit compiler gcc 4.5.2 (Ubuntu/Linux 4.5.2-8ubuntu4). Parallel execution of Bipartite, Tripartite and Proposed Algorithm with the help of fork() system call in Linux operating system.

V. CONCLUSIONS

This paper discusses various algorithms for modular multiplication of large numbers and evaluated them with respect to their accuracy, computation performance and efficiency. Each algorithm has its own features suitable for a specific field of application. No single algorithm provides a perfect solution to meet all demands; depending on the environment in which computation are to be performed, one algorithm may be preferable over another. Barrett and Montgomery Algorithm are efficient for smaller modular multiplication but for large modular multiplication tripartite and proposed Algorithm are efficient as shown in result.

The future work would be to use the Schönhage–Strassen algorithm for the multiplication step instead of Toom-Cook’s method in proposed algorithm.
REFERENCES


