Article

Accelerating the wheel factoring techniques

Alaa M. Zaki[1] , M. E. Bakr[2], Arwa M. Alsahangiti [3] and Saima Khan Khosa[4] and Khaled A. Fathy[5,*]

[1] Computer Science Department, Faculty of Science, Minia University, Minia 61519, Egypt; E-mail: alaa_zaki@mu.edu.eg
[2] Department of Statistics and Operations Research, College of Science, King Saud University, P.O. Box 2455, Riyadh 11451, Saudi Arabia; E-mail: mmibrahim@ksu.edu.sa
[3] Department of Statistics and Operations Research, College of Science, King Saud University, P.O. Box 2455, Riyadh 11451, Saudi Arabia; E-mail: Arwash@ksu.edu.sa
[4] Department of Mathematics and Statistics University of Saskatchewan, Saskatoon, SK, Canada; E-mail: skk807@mail.usask.ca
[5] Department of Mathematics, Faculty of Science, Al-Azhar University, Nasr City 11884, Egypt; E-mail: khaledfathy@azhar.edu.eg
* Correspondence: Khaled A. Fathy; khaled.fathy@su.edu.eg

Abstract: The efficiency with which an integer may be factored into its prime factors determines several public key cryptosystems′ security in use today. Although there is a quantum-based technique with a polynomial time for integer factoring, on a traditional computer, there is no polynomial time algorithm. We investigate how to enhance the wheel factoring technique in this paper. Current wheel factorization algorithms rely on a very restricted set of prime integers as a base. In this study, we intend to adapt this notion to rely on a greater number of prime integers, resulting in a considerable improvement in the execution time. The experiments on composite numbers n reveal that the proposed algorithm improves on the existing wheel factoring algorithm by about 75%.

Keywords: Integer factorization ; Wheel factorization; Cryptography

MSC: 11A41, 11A51

1. Introduction

The complexity of the integer factorization issue affects the security of several public key cryptosystems such as [11,20,23,25,36], while the exponentiation problem determines the effectiveness of such cryptosystems [9,10,34].

Algorithms for factoring a big odd composite number fall into two broad kinds, called special-purpose and general-purpose algorithms [5,16–18,26,30,31,35]. The special-purpose factoring algorithms [16] find tiny prime factors rapidly regardless of the value of n. If n has no tiny factors, employing one of the special-purpose factoring algorithms will almost surely fail, which is the primary issue with them. Examples of such factoring algorithms include the Fermat technique [5] with time complexity of $O(\sqrt{n})$, Pollard's elliptic curve method, Pollard's wheel factoring method, Pollard's p-method with time complexity of $O(\sqrt{n}\log n)$, and Pollard's $p-1$ method [15]. The general-purpose factorization techniques factor n regardless of the size of the prime factors, albeit they require exponential or subexponential time. Examples of such algorithms include the quadratic sieve with time complexity of $e^{(1+o(1))\sqrt{\ln n \ln\ln n}}$, continuous fraction method with time complexity of $O(e^{(\sqrt{2\log n \log\log n})})$, and number field sieve [18]. Recently, subexponential-time factoring methods utilizing binary decision and the Pisano period were proposed by Raddum, Varadharajan, and Wu [14] and Wu et al. [29], respectively. The number field sieve method with time complexity of $e^{(\frac{4}{\sqrt[3]{9}}+o(1))\sqrt{(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}}$ [28] has been shown to be the most effective method so far for factoring big n with large prime factors.

There are more factoring techniques, but they require more knowledge of cryptosystems or specific requirements for the prime factors (see [1,3,4,19]). Utilizing high-performance-Computing systems such as multicore systems [6,7,13,15], cloud computing systems [28], and graphics processing units [2,8] is one method for accelerating factoring algorithms.

In this paper, we focus on the challenge of factoring odd composite integers n. We investigate a new way to enhance the recent wheel factoring technique [33]. We expand the wheel method by using a greater number of prime integers as a basis rather than a limited number of prime integers.

The rest of the paper is organised as follows. In the second section, we will go over an overview of the most important works on the subject under consideration. The proposed algorithm will be presented in the third section. In the fourth section, we will provide a numerical illustration of the suggested method in action. The implementation of the algorithms will be presented in the fifth section. The results of the practical implementation of the algorithms will be examined in the sixth section. Finally, in the final section, we will conclude what we have accomplished.

2. Related Work

In this section, we will go over the three most important algorithms related to our research: Wheel Factorization Method (WFM), Forward Wheel Factorization Method (FWFM), and Backward Wheel Factorization Method (BWFM)[33].

2.1. Traditional Wheel factorization method (WFM)

In this subsection, we present a quick explanation of the WFM's central concept as well as its pseudocode for finding two factors for a composite integer. The WFM depends on picking the first k primes, known as the basis, say $B = \{b_1, b_2, ..., b_k\}$, where k is a tiny number, say 3 or 4. Then, WFM creates a list T, referred to as the turn or wheel, of integers that are coprime with every number in B. The product of the basis integers equals the circumference of the wheel (also known as primorial) $s = b_1 \times b_2 ... \times b_k$. Next, The integers in the wheel/turn are used by WFM to discover the lowest divisor of the integer to be factorised. To explain how WFM works to find a factor of n, B $= \{2, 3, 5\}$, and the circumference of a turn is $s = 30 = 2 \times 3 \times 5$. n must be divided by 2, 3, 5, and all other numbers that are congruent to 1, 7, 11, 13, 17, 19, 23, and 29, modulo 30, until a factor is discovered or $\sqrt{n}$ is reached. We begin by dividing n by 2, 3 and 5, and if it is not divisible by any of them, we go to the first iteration. In the first iteration, we divide n by 7, 11, 13, 17, 19, 23, 29 and 31, and if it is not divisible by any of them, we go to the second iteration. In the second iteration, we divide n by 37, 41, 43, 47, 49, 53, 59 and 61, and if it is not divisible by any of them, we go to the third iteration and so on. The difference between the matching items in any two successive turns is 30. The loop ends when we find a factor of n or reach a $\sqrt{n}$. If we reach the $\sqrt{n}$ without encountering a factor, then n is a prime number.

WFM does not have to begin from 1. For instance, we can start at 11, making the first turn set as follows: $\{11, 13, 17, 19, 23, 29, 31, 37\}$. In this situation, to begin, we must divide n by the elements of B and the number 7, i.e., 2, 3, 5, and 7. WFM checks whether n can be divided by the numbers in the following turn set, which are $\{41, 43, 47, 49, 53, 59, 61, 67\}$. The first 15 turn sets when WFM starts from 11 are shown in Table 1. The initial integer of the k^{th} turn is clearly equivalent to $init = 11 + 30(k - 1)$ and the following components of the k^{th} turn are $init + 2, init + 6, init + 8, init + 12, init + 18, init + 20$, and $init + 26$. The disparities between two consecutive integers in any turn are 2, 4, 2, 4, 6, 2, and 6, respectively, however, the difference between the last number in any round and the first integer in the next turn is 4. To handle all turns, these differences can be kept in an array, say inc. Algorithm 1 displays WFM pseudocode. As seen in Algorithm 2, the algorithm employs a function called FactorBasis to determine whether or not n has a factor from the basis set B.

Table 1. The first fifteen turns of circumference 30 starting from 11.

Turn no.	**init**	$init + 2$	$init + 6$	$init + 8$	$init + 12$	$init + 18$	$init + 20$	$init + 26$
1	11	13	17	19	23	29	31	37
2	41	43	47	49	53	59	61	67
3	71	73	77	79	83	89	91	97
4	101	103	107	109	113	119	121	127
5	131	133	137	139	143	149	151	157
6	161	163	167	169	173	179	181	187
7	191	193	197	199	203	209	211	217
8	221	223	227	229	233	239	241	247
9	251	253	257	259	263	269	271	277
10	281	283	287	289	293	299	301	307
11	311	313	317	319	323	329	331	337
12	341	343	347	349	353	359	361	367
13	371	373	377	379	383	389	391	397
14	401	403	407	409	413	419	421	427
15	431	433	437	439	443	449	451	457

Algorithm 1: WFM [33]

```
Input: A composite integer $m$.
Apply the FactorBasis function on m and store the returned result in $f$
Return $f$ and $\frac{m}{f}$ if the value of $f$ doesn't equal 1
Set $f = 11, i = 1$
Initialize the array $inc$ by $= \{2,4,2,4,6,2,6,4\}$
while $f \leq \lfloor\sqrt{m}\rfloor$ do
    if $m$ mod $f$ =0 then
        return $f$ and $\frac{m}{f}$
    else
        $f = f + inc[i]$
        if $i < 8$ then
            $i = i + 1$
        else
            $i = 1$
        end if
    end if
end while
Output: Two factors $f$ and $\frac{m}{f}$.
```

2.2. Forward WFM (FWFM)

In WFM, every while−loop iteration, Lines 6−17 of Algorithm 1 checks if m may be divided by the integer f. The value of f fluctuates with each turn depending on the value of $inc[j], 1 \leq j \leq 7$, Lines 10−15. The value of $inc[8]$ is used to establish the following turn′s start. With the FWFM, complete the while−loop in a single iteration rather than

Algorithm 2: FactorBasis [33]

```
1  Input: A composite integer $m$.
2  if $m \ mod \ 2 = 0$ return 2
3  if $m \ mod \ 3 = 0$ return 3
4  if $m \ mod \ 5 = 0$ return 5
5  if $m \ mod \ 7 = 0$ return 7
6  Otherwise return 1
7  Output: Either a prime number $f \in \{2, 3, 5, 7\}$ or 1.
```

eight separate iterations for each turn. There′s no reason to utilize the array inc because the idea can be accomplished by doing eight tests during every while−loop iteration. Additionally, there is no need to run the while−loop repeatedly while updating the counter i and testing its value. As a result, we can get rid of Lines 11 − 15 from the while−loop. Forward WFM (FWFM) is shown in Algorithm 3. FWFM tests each turn j of length 30 starting from $11 + 30j, j \geq 1$ using the subroutine TurnTest, Algorithm 4. If a factor of m is present, the function TurnTest returns it; otherwise, it returns 1. The function begins with f, the initial element of each round, and updates f by the numbers 2, 4, 2, 4, 6, 2, accordingly. Algorithms 3 and 4 present the complete pseudocodes for the FWFM and TurnTest algorithms, respectively.

Algorithm 3: Forward Wheel Factoring Method (FWFM) [33]

```
1  Input: A composite integer $m$.
2  Apply the FactorBasis function on $m$ and store the returned result in $f$
3  Return $f$ and $\frac{m}{f}$ if the value of $f$ doesn't equal 1
4  Set $i = 1$
5  while $i \leq \lfloor \sqrt{m} \rfloor$ do
6      Apply the TurnTest function on $m$ and $i$ and store the returned result in $f$
7      if $f \neq 1$ then
8          return $f$ and $\frac{m}{f}$
9      else
10         $i = i + 30$
11     end if
12 end while
13 Output: Two factors $f$ and $\frac{m}{f}$ of $m$.
```

2.3. Backward WFM (BWFM)

In several public-key cryptosystems, lowest prime factor is near to $[\sqrt{m}]$, like RSA [25], where the public modulus m is the product of two equal-sized prime numbers. Therefore, it is preferable to look for the factor between $[\sqrt{m}]$ and 2. (backward direction).

$nTurns = ([\sqrt{m}] - 10)/30$ is the total number of turns in the search space, counting from 11. The algorithm begins with the turn that was made at $i = 30(nTurns - 1) + 11$. The algorithm comes to an end if it discovers the factor of m. Otherwise, the algorithm searches in reverse order for a factor of m in the following turn by subtracting the circumference of turn 30 from i, *i.e.*, $i = i - 30$, This process is repeated for all turns. The algorithm looks for the factors 7, 5, 3, and 2 if the $nTurns$ iterations do not yield a factor. Algorithm 5 provides the whole pseudocode for the backward wheel factoring method (BWFM).

Algorithm 4: TurnTest [33]

```
Input: A composite integer $m$ and a positive integer $k$.
$f = 1$
if $m$ is divisible by $k$ then
    $f = k$
else
    increment k by 2
    if $m$ is divisible by $k$ then
        $f = k$
    else
        increment k by 4
        if $m$ is divisible by $k$ then
            $f = k$
        else
            increment k by 2
            if $m$ is divisible by $k$ then
                $f = k$
            else
                increment k by 4
                if $m$ is divisible by $k$ then
                    $f = k$
                else
                    increment k by 6
                    if $m$ is divisible by $k$ then
                        $f = k$
                    else
                        increment k by 2
                        if $m$ is divisible by $k$ then
                            $f = k$
                        else
                            increment k by 6
                            if $m$ is divisible by $k$ then
                                $f = k$
                        end if
                    end if
                end if
            end if
        end if
    end if
end if
return $f$.
Output: Either a factor $f$ of $m$ or 1.
```

Algorithm 5: Backward Wheel Factoring Method (BWFM) [33]

```
Input: A composite integer $m$.
$nTurns = \lceil \frac{\lfloor\sqrt{m}\rfloor - 10}{30} \rceil$
$i = 30 \times (nTurns - 1) + 11$
while $i \geq 11$ do
    $f = TurnTest(m, i)$
    if $f \neq 1$ then
        return $f$ and $\frac{m}{f}$
    else
        $i = i - 30$
    end if
end while
$f = FactorBasis(m)$
Output: Two factors $f$ and $\frac{m}{f}$ of $m$.
```

3. The Modified Wheel Factorization Method (MWFM)

The idea is based on the generalization of the WFM algorithm, which is based on an array *Base* of length 3 only. The main idea is to generalize the length of the Base array by reading lb of prime numbers from a file containing the first million prime numbers and storing them in the *Base* array, then calculating ws, which is the product of the *Base* array elements. The next step is to read all the primes up to ws, excluding the elements of the *Base* array, and store those elements in an array called *Turn*.
Using a function called NewFactorBasis, we begin the steps of the proposed algorithm by determining whether n is divisible by any of the elements of the *Base* array. We perform several iterations with a maximum of $\frac{\sqrt{n}}{ws}$ iterations in the absence of any factor of n in the *Base* array. We test whether n is divisible by any of the *Turn* array elements in each iteration. If there is no any factor of n within the *Turn* array, the *Turn* array elements are incremented by ws. This is accomplished through the use of a function known as NewTurnTest.
The full descriptions of NewFactorBasis, NewTurnTest and MWFM are shown in Algorithms 6,7 and 8.
NewFactorBasis algorithm runs in $O(1)$ because there are lb comparisons and divisions

Algorithm 6: NewFactorBasis

```
Input: A composite integer $n$ and an array $Base$ of the first $lb$ prime numbers
  $B = (b_0, b_1, \cdots, b_{lb-1})$
$p = 1$
for $i = 0$ to $lb - 1$ do
    if $n$ mod $b_i = 0$ then
        $p = b_i$
    else
    end if
end for
return $p$.
Output: Either a prime number from the array $Base$ or 1.
```

and lb does not depend on n. Also NewTurnTest algorithm runs in $O(1)$ because there are l comparisons and divisions and l does not depend on n. MWFM algorithm runs in $O(\sqrt{n})$ because the while loop in line 10 performs $\sqrt{n}$ iterations in the worst case.

Algorithm 7: NewTurnTest

```
Input: A composite integer $n$, an array $Turn$ of $l$ numbers $T = (t_0, t_1, t_2 \cdots, t_{l-1})$
  and an integer $ws$.
$f = 1$
for $i = 0$ to $l - 1$ do
    if $n$ mod $t_i$=0 then
        $f = t_i$
        break
    else
        $t_i = t_i + ws$
    end if
end for
return $f$.
Output: Either a factor $f$ from the array $Turn$ or 1.
```

Algorithm 8: Modified Wheel Factorization Method (MWFM)

```
Input: A composite integer $n$, the length of the base $lb$ and the file containing the
  1st million prime numbers.
Read the first $lb$ primes from the file and store them in an array B for the basis.
  $B = (b_0, b_1, \cdots, b_{lb-1})$
$ws = \prod_{i=0}^{lb-1} b_i$
Read the prime numbers up to $ws$ and store them in the $Turn$ array $T$ of length
  $l \simeq \frac{ws}{\ln ws}$ (The actual length is calculated during the reading process).
  $T = (t_0, t_1, t_2 \cdots, t_{l-1})$.
$p = NewFactorBasis(n, B, lb)$
if $p \neq 1$ then
    return $p$ and $\frac{n}{p}$
else
    $m = \sqrt{n}$
    while $t_0 \leq m$ do
        $p = NewTurnTest(n, T, l, ws)$
        if $p \neq 1$ then
            return $p$ and $\frac{n}{p}$
        end if
    end while
end if
return 1 and $n$
```

4. An Example

In this section, we give an example to show how MWFM algorithm works. Assume the worst case at which n is a prime number. Let $n = 505301$ and $bl = 4$
In line 2, four prime numbers are read and stored in B. $B = \{2, 3, 5, 7\}$.
In line 3, ws is computed. $ws = 210$.
In line 4, all prime number which are greater than 7 and up to 210 are read and stored in the array T. Also the actual length l of T is computed. $T = \{11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199\}$ and $l = 42$.
In line 5, we determine whether n is divisible by any of the elements of B by calling the function NewFactorBasis. n is not divisible by any of the elements of B. So NewFactorBasis returns 1 in p.
Because $p = 1$ we skip line 7 and execute from line 9.
In line 9, the square root of n is computed and stored in m. $m = 710$.
In lines 10-15, we perform several iterations depending on the value of t_0. In each iteration, NewTurnTest function is called to determine whether n is divisible by any of the current elements of T. If any factor p was found, the algorithm will return p and $\frac{n}{p}$ as in line 12-14.
If we do not find any factors, the elements of the array T will be modified by an increment of ws. The while loop will end when the value of t_0 exceeds $\sqrt{n} = 710$.
After the first iteration, we do not find any factor of n in T and the values of the elements are modified as follows:
$T = \{221, 223, 227, 229, 233, 239, 241, 247, 251, 253, 257, 263, 269, 271, 277, 281, 283, 289, 293, 299, 307, 311, 313, 317, 319, 323, 337, 341, 347, 349, 359, 361, 367, 373, 377, 383, 389, 391, 401, 403, 407, 409\}$.
After the second iteration, we do not find any factor of n in T and the values of the elements are modified as follows:
$T = \{431, 433, 437, 439, 443, 449, 451, 457, 461, 463, 467, 473, 479, 481, 487, 491, 493, 499, 503, 509, 517, 521, 523, 527, 529, 533, 547, 551, 557, 559, 569, 571, 577, 583, 587, 593, 599, 601, 611, 613, 617, 619\}$.
After the third iteration, we do not find any factor of n in T and the values of the elements are modified as follows:
$T = \{641, 643, 647, 649, 653, 659, 661, 667, 671, 673, 677, 683, 689, 691, 697, 701, 703, 709, 713, 719, 727, 731, 733, 737, 739, 743, 757, 761, 767, 769, 779, 781, 787, 793, 797, 803, 809, 811, 821, 823, 827, 829\}$.
Because the value of t_0 is still less than $\sqrt{n} = 710$, another iteration will be performed and T will be modified as follows:
$T = \{851, 853, 857, 859, 863, 869, 871, 877, 881, 883, 887, 893, 899, 901, 907, 911, 913, 919, 923, 929, 937, 941, 943, 947, 949, 953, 967, 971, 977, 979, 989, 991, 997, 1003, 1007, 1013, 1019, 1021, 1031, 1033, 1037, 1039\}$.
The condition of the while loop has been broken and we have reached line 17 and the value of p is still 1, so the algorithm will return 1 and $n = 505301$.

5. Experiments

This section describes the implementation of the proposed improvement on the best Known algorithm (BWFM). It is split into two subsections. The first subsection outlines the setup of the experimental investigation, which comprises the hardware, software, and data collection.
The average execution times of BWFM and MWFM are included in the second subsection.

5.1. Experimental Specifications

The studies were conducted out on a computer having a processor (Intel(R) Core(TM) $i7 - 4702MQ\ \ CPU\ \ @2.20GHz\ \ 2.20\ GHz$). Memory capacity of 8 GB.

Under Ubuntu 20.04 operating system, the algorithms are implemented using the C++ language and the OpenMP library [21], parallel loops are implemented using the OpenMP library. To operate huge numbers greater than 64 bits, the GMP (GNU Multiple Precision) library [12] is utilised.

The experimental study used composite numbers with sizes of 11-20 digits. The average time for 20 random numbers is the running time of each implemented algorithm for each fixed number of digits.

5.2. Execution Time

The running times of the algorithms BWFM and MWFM (which is considered the latest wheel factorization algorithms [33]) are displayed in this section. The average running times (in seconds) for BWFM and MWFM are shown in Table 2. The percentage of improvement for MWFM over BWFM is shown in Table 3 while Table 4 depicts the execution time of MWFM as the wheel size increases.

Table 2. Execution time (sec.) for BWFM and MWFM..

No. of digits	BWFM	MWFM
11	0.003	0.003
12	0.013	0.005
13	0.05	0.01
14	0.08	0.02
15	0.27	0.07
16	1.07	0.27
17	3.35	0.85
18	17.2	3.5
19	43.6	8.3
20	114	28.8

Table 3. Percentage of improvement of MWFM over BWFM.

No. of digits	Improvement
11	0%
12	62%
13	80%
14	75%
15	74%
16	75%
17	75%
18	80%
19	81%
20	75%
Average	75%

Table 4. The execution time of MWFM as the wheel size increases.

No. of digits	MWFM with wheel size:				
	4	5	6	7	8
18	15.7	9.2	6.2	4.6	3.5
19	31.3	17.8	12.5	8.8	8.3

6. Discussion

This section discusses and analyses the results of implementing the BWFM and MWFM algorithms on the data set in terms of running time, as shown in Tables 2, 3, and 4.

1. MWFM takes less time to execute than BWFM, except when the size equals 11 digits. See Table 2.

2. On average, the percentage of improvement for MWFM over BWFM is 75%. For all cases, the minimal and highest improvements are 0% and 81%, respectively. See Table 3.
3. The running time of MWFM decreases as the wheel size increases from 4 to 8. See Table 4.
4. The main disadvantage of MWFM is that it requires an array T, which will be rather huge if we use more than 8 prime integers as a basis.
5. Another flaw is that the method does not employ a theoretical foundation to identify the optimal number of prime integers to use as a basis.

7. Conclusion

In this paper, we addressed the integer factorization problem, which is one of the difficult problems associated with cryptanalysis of some public-key cryptosystems. The goal of integer factorization is to divide a composite number into prime factors. We have presented a new approach for improving the most recent wheel factoring method, BWFM. The approach is based on increasing the size of the Base array. The proposed algorithm performs significantly better when implemented on composite integers ranging in size from 11 to 20 digits. On average, MWFM outperforms BWFM by 75%. In the future, we will try to theoretically establish the ideal size of the Base array

Author Contributions: Software and Writing – original draft, Alaa M. Zaki, Khaled A. Fathy and M. E. Bakr; Writing – review and editing, Arwa M. Alsahangiti and Saima Khan Khosa. All authors have read and agreed to the published version of the manuscript.

Acknowledgments: This research was supported by King Saud University. Research Supporting Project number (RSPD-2023/538), King Saud University, Riyadh, Saudi Arabia

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Akchiche O., Khadir O. Factoring RSA moduli with primes sharing bits in the middle. *Appl Algebra Eng. Commun. Comput.* **2018**, *29(3)*, 245–259.
2. Atanassov E., Georgiev D., Manev N. Number theory algorithms on GPU clusters. *Modeling and Optimization in Science and Technologies, Springer* **2014**, *(2)*, 131–138.
3. Bahig HM., Nassr DI., Bhery A. Factoring RSA modulus with primes not necessarily sharing least significant bits.*Appl Math Inform Sci 2017*, *(11)*, 243 – 249.
4. Bahig HM., Nassr DI., Bhery A., Nitaj A. A unified method for private exponent attacks on RSA using lattices. *Int J. Found. Comput. Sci. 2020*, *31(2)*. 207 – 231.
5. Bahig HM., Mohammed A., Khaled A., AlGhadhban A., Bahig HM. Performance analysis of fermat factorization algorithms.*Int J Adv Comput Sci Appl2020*, *11(12)*, 340–350.
6. Bahig HM, Bahig HM, Kotb Y. Fermat factorization using a multi-core system. *Int. J. Adv. Comput. Sci. Appl.2020*, *11(4)*.
7. Brent RP. Some parallel algorithms for integer factorisation. *In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinfor-matics)1999*,*(1685)*, 1–22.
8. Durmus O., Cabuk UC., Dalkilic F. A study on the performance of base-m polynomial selection algorithm using GPU. *2020*.
9. Fathy K., Bahig H., Farag M. Speeding up multi- exponentiation algorithm on a multicore sys-tem. *J. Egypt Math. Soc. 2018*, *(26)*.
10. Fathy KA., Bahig HM., Ragab AA. A fast parallel modular exponentiation algorithm. *Arab. J. Sci. Eng.2018*, *(43)*.
11. Fujioka A., Suzuki K., Xagawa K., Yoneyama K. Strongly secure authenticated key exchange from factoring, codes, and lat-tices.*Design Codes Cryptogr2015*,*(76)*, 469–504.
12. GMP. library, GNU multiple precision arithmetic library. https:// gmplib. org/.
13. Gulida KR., Ultanov IR. Comparative analysis of integer factorization algorithms using CPU and GPU. *MANAS J Eng 2017*, *(5)*, 53–63.
14. Varadharajan S., Raddum H. Factorization using binary decision diagrams. *Cryptogr Com−mun2019*, *(11)*, 443–460.
15. Koundinya AK., Harish G., Srinath NK., Raghavendra GE., Pramod YV., Sandeep R., Punith KG. Performance analysis of parallel pollard′s RHO factoring algorithm. *Int. J. Comput. Sci. Inform. Technol. 2013*, *(5)*.
16. Lenstra AK. Integer factoring. *Designs Codes Cryptogr. 2000*, *(19)*, 101–128.
17. Menezes AJ., Katz J., van Oorschot PC., Vanstone SA. Handbook of Applied Cryptography. *CRC Press1996*.
18. Montgomery PL. A survey of modern integer factorization algorithms. *CWI Quart1994*, *(7)*, 337–366.

19. Nassr DI., Bahig HM., Bhery A., Daoud SS. A new rsa vulnerability using continued fractions. *In AICCSA 08 - 6th IEEE/ACS International Conference on Computer Systems and Applications2008*, 694–701.
20. Nimbalkar AB.. The digital signature schemes based on two hard problems: factorization and discrete logarithm.*In: In: Bokhari M, Agrawal N, Saini D (eds) Cyber Security 2018*, *(729)*, 493–498.
21. OpenMP. https://www.openmp.org/
22. Peng WC., Wang BN., Hu F., Wang YJ., Fang XJ., Chen XY., Wang C. Factoring larger integers with fewer qubits via quantum annealing with optimized parameters.*Sci. China Phys. Mech. Astron. 2019*, *(62)*,
23. Poulakis D. A public key encryption scheme based on factoring and discrete logarithm. *J. Discrete Math. Sci. Cryptogr. 2009*, *(12)*, 745–452.
24. Pritchard P. Explaining the wheel sieve. *Acta Inform. 1982*, *(17)*, 477–485.
25. Rivest RL., Shamir A., Adleman LM. A method for obtaining digital signatures and public key cryptosystems.*Commun ACM1987*, 120–126.
26. Rubinstein-Salzedo S. Clever factorization algorithms and primality testing. *2018*.
27. Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer.*SIAM J. Comput. 1997*, *(26)*, 1484–1509.
28. Valenta L., Cohney S., Liao A., Fried J., Bodduluri S., Heninger N. Factoring as a service. *In: Grossklags J., Preneel B. (eds) Financial cryptography and data security. 2017*, *(6903)*, 321–338.
29. Wu L., Cai HJ., Gong Z. The integer factorization algorithm with pisano period. *IEEE Access 2019*, *(7)*, 167250–167259.
30. Yan SY. Primality testing and integer factorization in public-key cryptography. *advances in information security. Springer 2009* ,*(11)*.
31. Yan SY. Factoring Based Cryptography.*2019*, 217–286.
32. Yan SY., James G. Can integer factorization be in p?. *In: International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA′06) 2006*, 266–266.
33. Bahig HM., Nassr DI., Mohammed A., Khaled A., Bahig HM. Speeding up wheel factoring method. *The Journal of Supercomputing2022*.
34. Bahig, H.M.; Nassr, D.I.; Mahdi, M.A.; Bahig, H.M. Small Private Exponent Attacks on RSA Using Continued Fractions and Multicore Systems. *Symmetry 2022*, 14, 1897. https://doi.org/10.3390/sym14091897
35. Hazem M. Bahig, Hatem M. Bahig and Yasser Kotb, Fermat Factorization using a Multi-Core System. *International Journal of Advanced Computer Science and Applications(IJACSA)*, *2020*, 11(4). http://dx.doi.org/10.14569/IJACSA.2020.0110444
36. Zheng, M. Revisiting the Polynomial-Time Equivalence of Computing the CRT-RSA Secret Key and Factoring. *Mathematics*, *2022*, 10, 2238. https://doi.org/10.3390/math10132238