Asian Journal of Research in Computer Science

Volume 17, Issue 6, Page 85-95, 2024; Article no.AJRCOS.115573
ISSN: 2581-8260

Factorization Algorithm for Semi-primes and the Cryptanalysis of Rivest-Shamir-Adleman (RSA) Cryptography

Richard Omollo [a,b*] and Arnold Okoth [a,b]

[a] Department of Computer Science and Software Engineering, School of Informatics and Innovative Systems, Kenya.
[b] Jaramogi Oginga Odinga University of Science and Technology, Box 210-40601, Bondo, Kenya.

Authors' contributions

This work was carried out in collaboration between both authors. Both authors read and approved the final manuscript.

Article Information

DOI: 10.9734/AJRCOS/2024/v17i6458

Open Peer Review History:
This journal follows the Advanced Open Peer Review policy. Identity of the Reviewers, Editor(s) and additional Reviewers, peer review comments, different versions of the manuscript, comments of the editors, etc are available here: https://www.sdiarticle5.com/review-history/115573

Original Research Article

Received: 12/02/2024
Accepted: 16/04/2024
Published: 18/04/2024

ABSTRACT

This paper introduces a new factoring algorithm called Anorld's Factorization Algorithm that utilizes semi-prime numbers and their implications for the cryptanalysis of the Rivest-Shamir-Adleman (RSA) cryptosystem. While using the concepts of number theory and algorithmic design, we advance a novel approach that notably enhances the efficiency of factoring large semi-prime numbers compared to other algorithms that have been developed earlier. In our approach, we propose a three-step algorithm that factorizes relatively large semi-primes in polynomial time. We have introduced factorization up to 12-digit semi-prime using Wolfram|Alpha, a mathematical software suitable for exploring polynomials. Additionally, we have discussed the implications of the new algorithm for the security of RSA-based cryptosystems. In conclusion, our research work

**Corresponding author: E-mail: comolor@hotmail.com;*

emphasizes the important role of factoring algorithms in the cryptanalysis of RSA cryptosystems and proposes a novel approach that bolsters the efficiency and effectiveness of semi-prime factorization, thereby informing the development of more powerful cryptographic protocols.

Keywords: Arnold's Factorization Algorithm (A.F.A.); modular factorial; euclidean algorithm; RSA cryptography.

1. INTRODUCTION

RSA Cryptography was officially introduced in 1978 by Ronald Rivest, Adi Shamir, and Leonard Adleman of MIT [1-2]. Its efficiency is based on the difficulty of factorizing very large semi-primes. Over 40 years later since its inception, there have been several attempts to find efficient algorithms to factorize these large semi-primes [3]. The Shor Algorithm which was developed by Peter Shor in 1994 is the only algorithm that is currently capable of breaking RSA Cryptography in polynomial time. However, the Shor Algorithm is a quantum algorithm that can only work efficiently on a quantum computer with a sufficient number of quantum bits. Furthermore, the quantum computer should be able to perform its operation without interference from quantum noise. The development of such a quantum computer is still speculated to be years away. However, there have been significant improvements in the development of quantum computers over the years [4-8]. Recently, Google claimed to have reached 'quantum supremacy' by performing a mathematical calculation in just 220 seconds that would otherwise take the best supercomputers thousands of years to calculate [9]. IBM released a statement later in the same week claiming that their supercomputer could perform the same calculation in just 1.5 days. For classical computers, the fastest known algorithm to factorize very large integers with over 100 digits is known as the General Number Field Sieve.

Arnold's Factorization Algorithm [3] is a simple three-step technique that specifically factorizes semi-primes. The first step involves getting the square root of the semi-prime (N). The second step is getting the double factorial of your result in step one. Then the final step involves finding the GCD of your result in step two with the semi-prime (N) using the Euclidean Algorithm [10-13]. The general Arnold's Factorization Algorithm is

$$GCD(a!!, N) \quad \text{Eqn. 1}$$

Implementing the algorithm step by step is not feasible for semi-primes with more than 16 digits. All the semi-primes with 16 digits and below can be factorized in just a matter of seconds using powerful online mathematical software such as Wolfram|Alpha [14-15]. This is independent of the in-built factorization algorithm that this software already uses. By combining the second step of Arnold's Factorization Algorithm and the first step of the Euclidean Algorithm, we reduce the integer factorization problem to a modular factorial problem. This gives us several options to execute the second step without obtaining an integer overflow through the double factorial function. Crypt-analysis of RSA Cryptography was also done in the second step of the algorithm. This was done using the gamma function instead of the double factorial function taking into consideration the time and space complexity [16].

There are generally several algorithms each with different approaches to factorizing integers [17-20]. Arnold's Factorization Algorithm targets specifically semi-primes. This is because the implications of factorizing semi-primes are much greater in internet security through RSA digital signatures [21-23] and the Diffie-Hellman key agreement protocol [24-25]. The Arnold Factorization Algorithm aims at breaking down a hard mathematical problem, integer factorization, into feasible polynomial steps and/or procedures.

2. METHODOLOGY

In this section, we established fundamental definitions with illustrative examples and outlined essential procedures pertinent to the research methodology.

Definition 2.1

Arnold's Factorization Algorithm (A.F.A.) is a simple three-step algorithm for factorizing semi-primes [3].

Definition 2.2

Factorial [26-27] refers to the multiplication of all the numbers from one up to the number in which we are obtaining its factorial. It is usually denoted by an exclamation mark (!).

For example:

1! = 1
2! = 1×2 =2
3! = 1 × 2× 3 = 6
4! = 1× 2 × 3 × 4 = 24
5! = 1 × 2 × 3 ×4 × 5 = 120
6! = 1 × 2 × 3 × 4 × 5 × 6 = 720
7! = 1× 2 ×3 × 4× 5 × 6 × 7 = 5040
8! = 1× 2 × 3 × 4 × 5 × 6 × 7 × 8 = 40320

Definition 2.3

The Double Factorial of an integer (n) [26-27] refers to the multiplication of all the numbers from one up to (n) which have the same parity, either even or odd. It is usually denoted by double exclamation marks (n!!). It is also referred to as semi-factorial.

For example:

1!! = 1
2!! = 2
3!! = 1 × 3 =3
4!! = 2 × 4 =8
5!! = 1 × 3 × 5 = 15
6!! = 2 × 4 × 6 = 48
7!! = 1 × 3 × 5 × 7 = 105
8!! = 2 × 4 × 6 × 8 = 384

We notice that the double factorial of a number like 6 is significantly smaller than the factorial of 6.

Definition 2.4

Modular Factorial [28] simply refers to the remainder after dividing a factorial by another integer.

For example:

5! mod 6 = 0
5!! mod 6 = 3
6! mod 7 = 6
6!! mod 7 = 6

Definition 2.5

Quantum supremacy [29] refers to the state in which a quantum computer can perform a programmable operation that no classical computer can do in a feasible amount of time. It is also referred to as quantum advantage.

Definition 2.6

Crypt-analysis [30] refers to the tactical and mathematical analysis of ciphers and cryptosystems to find how they work so that you can exploit their weaknesses and eventually break them.

Definition 2.7

Modular Exponentiation [31] is just exponentiation calculated over modulus. It is very crucial in public-key cryptography.

For example:

$7^3 \bmod 5 = 3$
$7^3 = 343$
$343 \bmod 5 = 3$

Where 3 is the exponent.

Definition 2.8

Semi-Primes [3, 32-35] refers to the product of multiplying two prime numbers.

For example:

5 × 7 = 35 is the semi prime
101 × 103 = 10403 is the semi prime

Definition 2.9

Time and Space Complexity [36]: Time complexity refers to the total number of operations that a particular algorithm does to perform its task. It is the computational time taken to run an algorithm and is denoted by the big *O* notation. Space Complexity of a computer algorithm refers to the total space that is required by that algorithm to run on a computer.

Definition 2.10

Cipher text [37] refers to an encrypted message that cannot be read unless it is converted back to plain text using an algorithm.

Definition 2.11

RSA numbers [38]: These are semi-primes used for encryption in RSA Cryptography. The two prime numbers used are relatively of the same size.

3. RESULTS AND DISCUSSION

In this section, we explored the findings and outcomes derived from the study, providing a comprehensive analysis and interpretation of the results obtained.

3.1 Arnold's Factorization Algorithm

Below are the three steps for factorizing semi-primes using Arnold Factorization Algorithm. We are going to use the semi-primes 55, 65, and 77 throughout our examples.

Step I: Square root

1. Get the square root of a semi-prime (N). This separates the two prime numbers of the semi-prime with the smallest prime number being smaller than the square root.

2. Round it off to the nearest whole number. If the result is an even number subtract 1 to get an odd number. (The reason for this operation is that we are performing a double factorial to the number in the second step). It also has to be odd because all the prime numbers apart from 2 are odd.

3. The result from the above procedures is denoted by the letter (a). If the semi-prime is a square of a prime number, then the factorization stops at step 1.

For example:

To factorize the semi-prime 55
Square root: √55 = 7.416198487…
Round off to the nearest whole number:
7.416198487 gives us 7
7 is an odd number so it becomes our (a)

To factorize the semi-prime 65
Square root: √65 = 8.062257748...
Round off to the nearest whole number:
8.062257748 gives us 8
8 is an even number so we subtract 1: 8 – 1 = 7
7 is an odd number so it becomes our (a)

To factorize the semi-prime 77
Square root: √77 = 8.77496438…
Round off to the nearest whole number:
8.77496438 gives us 9
9 is an odd number so it becomes our (a)

Therefore (a) can be defined by the following equation:

$$a = \lceil \sqrt{N} \rceil \qquad \text{Eqn. 2}$$

Which simply means that (a) is equal to the smallest integer equal or greater than $\sqrt{N}$. However, it is an even number you subtract one to get an odd number which might be smaller than the case

Step II: Double Factorial/Gamma Function

Since the value of (a) is odd we get its double factorial. That is, we multiply all the odd numbers from one up to (a). We can also get the factorial but the value is much larger. In this step, the values can become astronomically large for larger semi-primes. In fact, in most personal computers getting the double factorial of a 9-digit number gives an integer overflow error. We discussed this in later section on how to curb this computational space complexity problem. The result from step 2 is denoted by (a!!). Factorial can also be expressed as a gamma function. For our case a! = Γ (a + 1). Where the gamma function is denoted by Γ.

For example:

To factorize the semi-prime 55

We have seen from the above examples that the value (a) for the semi-prime 55 is 7.

Double factorial: 7!! = 1 × 3 × 5 × 7 = 105

Alternatively, the factorial: 7! = 1× 2 ×3 × 4 × 5 × 6 × 7 = 5040

So, our (a!!) for 55 is 105, and (a!) is 5040

Both the double factorial and factorial will give you the correct answer after the third and final step.

To factorize the semi-prime 65

We saw from step 1 that the value (a) for the semi-prime 65 is also 7.

Double factorial: 7!! = 1 × 3 × 5 × 7 = 105

Alternatively, the factorial: 7! = 1× 2 ×3 × 4 × 5 × 6 × 7 = 5040

So, our (a!!) for 55 is 105, and (a!) is 5040

We notice we get the same values for both 55 and 65. This is because of the proximity of their square root.

To factorize the semi-prime 77

From the first step, the value (a) for the semi-prime 77 is 9.

Double factorial: 7!! = 1 × 3 × 5 × 7 × 9 = 945

Alternatively, the factorial: 7! = 1× 2 ×3 × 4 × 5 × 6 × 7 × 8 × 9 = 362880

So, our (a!!) for 77 is 945, and (a!) is 362880.

We observe that it is only logical to use double factorial not only because it gives us relatively smaller values but also because prime numbers are odd numbers.

Step III: GCD using Euclidean Algorithm

Here we get the GCD of the double factorial (a!!) and the semi-prime (N) using the Euclidean Algorithm. This is denoted by the expression

GCD (a!!, N)

The GCD of the expression above gives the smallest prime number. The first step of calculating the Euclidean Algorithm involves getting modular factorial. That is a!! mod N = r . Then N mod r = s is computed. This procedure is repeated until the mod is 0. The mod before 0 is the GCD and the smallest prime number for the semi-prime N. No matter how big a!! is, after getting the modular factorial r < N.

For example:

To factorize the semi-prime 55

We saw that a!! for 55 is 7!! = 105

GCD (Euclidean Algorithm): GCD (7!!, 55) or GCD (105, 55).

105 mod 55 = 50
55 mod 50 = **5**
50 mod 5 = 0

The mod before 0 is 5. Therefore, the GCD (7!!, 55) = 5

Therefore, the smallest prime number is 5.

Alternatively, we can use factorial instead of double factorial.

GCD (Euclidean Algorithm): GCD (7!, 55) or GCD (5040, 55).

5040 mod 55 = 35
55 mod 35 = 20
35 mod 20 = 15
20 mod 15 = 5
15 mod 5 = 0

The mod before 0 is 5. Therefore the GCD (7!, 55) = 5. The smallest prime number is also 5. This still gives you the same answer as getting the GCD of the double factorial.

To factorize the semi-prime 65

We saw that a!! for 65 is 7!! = 105

GCD (Euclidean Algorithm): GCD (7!!, 65) or GCD (105, 65).

105 mod 65 = 40
65 mod 40 = 25
40 mod 25 = 15
25 mod 15 = 10
15 mod 10 = **5**
10 mod 5 = 0

The mod before 0 is 5. Therefore, the GCD (7!!, 65) = 5

The smallest prime number is 5.

Alternatively: GCD (Euclidean Algorithm): GCD (7!, 65) or GCD (5040, 65).

5040 mod 65 = 35
65 mod 35 = 30
35 mod 15 = **5**
30 mod 5 = 0

The mod before 0 is 5. Therefore the GCD (7!, 65) = 5. The smallest prime number is also 5.

To factorize the semi-prime 77

We saw that a!! for 77 is 9!! = 945

GCD (Euclidean Algorithm): GCD (9!!, 77) or GCD (945, 77).

945 mod 77 = 21
77 mod 21 = 14
21 mod 14 = **7**
14 mod 7 = 0

The mod before 0 is 7. Therefore, the GCD (9!!, 77) = 7

The smallest prime number is 5.

Alternatively, we can use factorial instead of double factorial.

GCD (Euclidean Algorithm): GCD (9!, 77) or GCD (362880, 77).

362880 mod 77 = 56
77 mod 56 = 21
56 mod 21 = 14
21 mod 14 = **7**
14 mod 7 = 0

The mod before 0 is 7. Therefore the GCD (9!, 77) = 7.

The smallest prime number is also 7.

That is how we factorize semi-primes using Arnold's Factorization Algorithm. The second and third steps of the algorithm can be applied to online mathematical software such as Wolfram|alpha. This enables factorization of semi-primes of up to 12 digits in just a matter of seconds.

For example:

To factorize the semi-prime 300293215969. (12 digit semi-prime)

Step 1: Square root: √300293215969 = 547990.1605…

Round off: 547990.1605 gives us 547990

547990 is an even number so we subtract 1 to get 547989

So, our (a) value for the semi-prime 300293215969 is 547989.

This step is implemented on a Python code shown below:

```
import math

semiprime=eval(input('semiprime'))
k=math.isqrt(int(semiprime))
print(k)
f=len(str(k))
print(f)
```

This code finds the integral values of the square root of an integer with over 1000-digit numbers in just seconds. That is, it truncates the values after the decimal point and only prints the whole number.

Steps II and III: This is performed on the Wolfram|alpha online software. The digits are fed in the form of the expression GCD (a!!, N). From our calculation the value of a = 547989 and N = 300293215969 (Semi-prime). The result is 317617 which is the GCD and also the smallest prime number that divides 300293215969.

3.2 Optimization of Arnold's Factorization Algorithm

In this section, we dedicated our discussion to how to make Arnold's Factorization Algorithm more efficient. We considered several ways in which the algorithm can be implemented to avoid integer overflow errors.

3.2.1 Modular arithmetic

One of the properties of modular arithmetic states that;

$$[(a \bmod N) * (b \bmod N) * (c \bmod N)] \bmod n = (a * b * c) \bmod N \quad \text{Eqn 3}$$

Where N is the semi-prime in our case. The primitive method of implementing Arnold's Factorization Algorithm is by applying the right side of the above equation. It is relatively fast because of the factorial function however, it is more likely to give an integer overflow error for very large semi primes. Therefore, it is recommended to use the left side of the equation when writing the algorithm for relatively large semi-primes. This means that the result after each iteration will not be bigger than the semi-prime N.

For example, where;

a = 7
b = 37
c = 43
N = 35

Solution (Primitive Method)

(a * b * c) mod N
(7 * 37 * 43) mod 35 = (11137) mod 35 = 7

Note that if the values of a, b, and c were large the above method would give an integer overflow error when run on a personal computer.

(Optimized Algorithm Method)

[(a mod N) * (b mod N) * (c mod N)] mod N

[(7 mod 35) * (37 mod 35) * (43 mod 35)] mod 35 = [(7) * (2) * (8)] mod 35 = 7

(112) mod 35 = 7

Alternatively

(((a mod N) * (b mod N)) mod N * (c mod N) mod N)

(((7 mod 35) * (37 mod 35)) mod 35 * (43 mod 35) mod 35) which can also be written in the following sequence. The result of an initial step is the start of the subsequent step.

1. 7 mod 35 = 7
2. 7 * (37 mod 35) mod 35 = (7 * 2) mod 35 = 14
3. 14 * (43 mod 35) mod 35 = (14 * 8) mod 35 = 7

Both methods give the same answer however the optimized algorithm method gives a smaller number of which the result after an operation is always smaller than the semi-prime. This is seen in the alternative of the Optimized Algorithm Method. This is very crucial in the implementation of the second and third steps of the AFA.

3.2.2 Sequencing and gamma function

In this section, we optimize the second step of AFA. Both factorial and double factorial involve multiplying numbers with a common difference between them. Thus, we apply properties of arithmetic sequencing or progression.

For example:

(Factorial)

5! = 1×2×3×4×5 = 120

Here we see that the sequence is 1, 2, 3, 4, and 5 with a common difference of 1. The first term is 1 and the number of terms is 5. In other words, factorial simply means getting the product of the arithmetic sequence above.

(Double Factorial)

5!! = 1×3×5 = 15

In this case, the common difference is 2. The number of terms is 3 and the first term is 1.

The formula for getting the product of an arithmetic sequence is as follows:

$$P = d^n \times \frac{\Gamma\left(\frac{a1}{d}+n\right)}{\Gamma\left(\frac{a1}{d}\right)} \qquad \text{Eqn 4}$$

Where:

d is the common difference.
n is the number of terms.
a1 is the value of the first term.
Γ is the gamma function.
P is the product of the arithmetic sequence.

We now embark on the properties of the gamma function which are critical in our crypt-analysis process.

They include:

1. $\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$;
2. $\Gamma(\alpha+1) = \alpha\Gamma(\alpha)$;
3. $\Gamma(b) = (b+1)!$

In the example below we look at the application of double factorial, gamma function, and sequencing as an optimization of AFA.

Example:

Given:
d = 2
a1 = 1
n = 3

Which is the sequence 1, 3, 5.

Using the formula for getting the product of an arithmetic sequence we get:

$$P = 2^3 \times \frac{\Gamma\left(\frac{1}{2}+3\right)}{\Gamma\left(\frac{1}{2}\right)}$$

Adding the numbers in brackets we get:

$$P = 8 \times \frac{\Gamma\left(\frac{7}{2}\right)}{\Gamma\left(\frac{1}{2}\right)}$$

From the first property of the gamma function we see that the denominator becomes $\sqrt{\pi}$ therefore our equation becomes:

$$P = 8 \times \frac{\Gamma\left(\frac{7}{2}\right)}{\sqrt{\pi}}$$

From the second property of the gamma function we can be able to re-write the numerator as follows:

$$\Gamma\left(\frac{7}{2}\right) = \Gamma\left(\frac{5}{2}+1\right)$$

$$= \frac{5}{2}.\Gamma\left(\frac{5}{2}\right)$$
$$= \frac{5}{2}.\Gamma\left(\frac{3}{2}+1\right)$$
$$= \frac{5}{2}.\frac{3}{2}.\Gamma\left(\frac{3}{2}\right)$$
$$= \frac{5}{2}.\frac{3}{2}.\Gamma\left(\frac{1}{2}+1\right)$$
$$= \frac{5}{2}.\frac{3}{2}.\frac{1}{2}.\Gamma\left(\frac{1}{2}\right)$$

Using the first property we get:

$$= \frac{5}{2}.\frac{3}{2}.\frac{1}{2}.\sqrt{\pi}$$
$$= \frac{15}{8}.\sqrt{\pi}$$

The whole equation now becomes:

$$P = 8 \times \frac{\frac{15}{8}.\sqrt{\pi}}{\sqrt{\pi}}$$

This can be simplified to get:

$$P = 8 \times \frac{15}{8}$$
$$P = 15$$

Therefore, the product of the sequence 1, 3, 5 is 15.

We have proved that the formula of getting the product of an arithmetic sequence can be simplified by using gamma properties.

We can also derive the double factorial of the value of *a* using the formula for getting the product of an arithmetic sequence. Since it is obtaining the double factorial, the following values are always constant.

a1 = 1
d = 2

P will be the double factorial of a

Replacing these values on the arithmetic sequence product formula we get the following formula:

$$(a)!! = d^n \times \frac{\Gamma\left(\frac{1}{2}+n\right)}{\Gamma\left(\frac{1}{2}\right)} \qquad (Eqn\ 5)$$

Where $n > 1$.

Remember that $a = \lceil \sqrt{N} \rceil$

Where N is the semi-prime and the value of (a) is a whole odd number. It is obtained by getting the square root of N and rounding it off to the nearest whole number. If it is even, 1 is added to get an odd number which becomes our value for *a*. We thus obtain our value for *n* from the equation below.

$$n = \frac{a+1}{2} \qquad (Eqn\ 6)$$

This can also be written as:

$$a = 2n - 1 \qquad (Eqn\ 7)$$

Replacing the value of a in (equation i) we obtain:

$$(2n-1)!! = d^n \times \frac{\Gamma\left(\frac{1}{2}+n\right)}{\Gamma\left(\frac{1}{2}\right)} \qquad \text{(Eqn 8)}$$

Therefore, the formula of getting the double factorial using the number of terms now becomes:

$$(2n-1)!! \qquad \text{(Eqn 9)}$$

4. CRYPT-ANALYSIS OF RSA CRYPTOSYSTEM USING ARNOLD'S FACTORIZATION ALGORITHM

In this section, our main focus was on what renders the factorization of semi-primes pivotal in the cryptanalysis of the RSA cryptosystem. The significance was apparent from the capability to decrypt ciphertexts speedily upon obtaining one of the prime numbers employed in encryption. Moreover, we explored the salient properties inherent in RSA numbers used for encryption. These include:

1. The two prime numbers used are relative of the same size (number of digits).

2. The prime numbers are not too close to each other otherwise they can be easily factorized using Fermat's Factorization Method [39].

3. The RSA numbers are very large being at least 300 digits and above.

4. The prime numbers are normally half the size of the RSA numbers.

Since the prime numbers are large and are relatively of the same size, we didn't start multiplying numbers from one up to the square

root of that number. For illustration purposes, we used the semi-prime 300293215969 we had used earlier as our RSA number.

Example:

Crack the RSA number 300293215969. A 12-digit semi-prime.

By applying the first step of AFA we are first going to get the square root of the RSA number.

Step 1: Square root: √300293215969 = 547990.1605…

Round off: 547990.1605 gives us 547990

547990 is an even number so we subtract 1 to get 547989

So, our (a) value for the semi-prime 300293215969 is 547989.

Step 2: Using the fourth property we know that the prime numbers have half the number of digits as the RSA numbers except ±1 digit always. Therefore, instead of getting the double factorial of 547989, we used the Eqn 4. This means that the first term should be the least odd number with 6 digits since the semi-prime has 12 digits.

Our equation is as follows:

$$a!! = 2^{273995} \times \frac{\Gamma\left(\frac{100001}{2} + 273995\right)}{\Gamma(\frac{100001}{2})}$$

Where:

d = 2
a = 547989
$n = \frac{547989+1}{2} = 273995$
a1 = 100001
N = 300293215969

Step 3: We now apply the Euclidean Algorithm to obtain the GCD (a!!, N).

$$\left(2^{273995} \times \frac{\Gamma\left(\frac{100001}{2}+273995\right)}{\Gamma\left(\frac{100001}{2}\right)}\right) mod\ 300293215969 = 210086811799$$

$300293215969 \bmod 210086811799 = 90206404170$
$210086811799 \bmod 90206404170 = 29674003459$
$90206404170\ mod\ 29674003459 = 1184393793$
$29674003459\ mod\ 1184393793 = 64158634$
$1184393793\ mod\ 64158634 = 29538381$
$64158634\ mod\ 29538381 = 5081872$
$29538381\ mod\ 5081872 = 4129021$
$5081872\ mod\ 4129021 = 952851$
$4129021\ mod\ 952851 = \mathbf{317617}$
$952851\ mod\ 317617 = 0$

The mod before 0 is 317617 which is the smallest prime number and factor of the semi-prime 300293215969.

Dividing 300293215969 by 317617 we get 945457 which is the other prime factor.

As we can see both the prime numbers have six digits and they are relatively not too close to each other.

It is important to note that AFA optimizes multiplication in comparison to the traditional trial division (brute force) which takes a relatively longer time. This is because multiplication falls under polynomial problems.

5. CONCLUSION

In this paper, we have demonstrated that the Arnold Factorization Algorithm has the potential to optimize the factorization process of considerably large semi-prime numbers. We argue confidently that it effectively mitigates integer overflow errors and significantly reduces computational time. The effects of a fully optimized Arnold Factorization Algorithm expand far beyond just algorithmic efficiency, pervading both the realms of Mathematics and Computer Science. Consequently, there arises an imperative to protect existing cryptosystems, thereby guaranteeing enhanced security measures across various digital platforms including e-commerce websites, email communication, virtual private networks (VPNs), online banking platforms, and the HTTPS protocol for web browsers.

Additionally, even in the theoretical case where quantum computers capable of factorizing such large semi-primes become apparent, the viability and cost-effectiveness of the Arnold Factorization Algorithm remain exceptional. Given its capacity to run on personal computers, the Arnold Factorization Algorithm stands as a resilient and accessible solution, offering a robust defense against potential cryptographic vulnerabilities posed by quantum computing advancements.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

1. Sattar J Aboud, Mohammad A AL-Fayoumi, Mustafa Al-Fayoumi, Haidar S Jabbar. An efficient RSA public key encryption scheme. Fifth international conference on information technology: New generations. IEEE Computer Society; 2008.
2. Michael NJ, Ogoegbulem O, Obukohwo V, Egbogho HE. Number Theory in RSA Encryption Systems. GPH – International Journal of Mathematics. 2023;6:11.
3. Richard Omollo, Arnold Okoth. Large Semi Primes Factorization with Its Implications to RSA Cryptosystems. BOHR International Journal of Smart Computing and Information Technology. 2022;3(1):1–8.
4. Douglas R, Stinson, Maura B. Paterson. Cryptography Theory and Practice 4th Edition. CRC Press; 2019.
5. Dirk B, Artur E, Anton Z. The Physics of Quantum Information. XIV – 315, Springer; 2000.
6. Matthew H. Quantum Computing and Shor's Algorithm. 2015;1-7.
7. Adjie Wahyu Wicaksono, Arya Wicaksana. Implementation of Shor's quantum factoring algorithm using project Q framework. International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249-8958 (Online). 2019; 8(6S3).
8. Arun Bhalla, Kenneth Eguro, Matthew Hayward. Quantum Computing, Shor's Algorithm, and Parallelism; 2015.
9. Gil Kalai, Yosef Rinott, Tomer Shoham. Google's 2019 Quantum Supremacy Claims: Data, Documentation, and Discussion; 2022. Available:https://www.arxiv.org
10. Jeffrey Shallit. Origins of the Analysis of the Euclidean Algorithm. Historia Mathematica. 1994;21:401-419.
11. Carlos M Falcon Rodriguez, Maria A. Garcia Cruz, Claudia Falcon. Full Euclidean Algorithm by Means of a Steady Walk. Applied Mathematics. 2021;12: 269-279
12. Anton Iliev, Nikolay Kyurkchiev, Asen Rahnev. A New Improvement Euclidean Algorithm for Greatest Common Divisor. I. Neural, Parallel, and Scientific Computations. 2018;26(3):355-362. ISSN: 1061-5369
13. Franz Lemmermeyer. The Euclidean Algorithm in Algebraic Number Fields. Mathematics Subject Classification; 2004.
14. Wan Nur Shaziayani Wan Mohd Rosly, Sharifah Sarimah Syed Abdullah, Fuziatul Norsyiha Ahmad Shukri. The uses of Wolfram Alpha in Mathematics. Articles of Teaching and Learning in Higher Education. 2020;1:96 – 103.
15. Hiyam, Bataineh, Ali Zoubi, Abdalla Khataybeh. Utilizing MATHEMATICA Software to Improve Students' Problem Solving Skills of Derivative and its Applications. International Journal of Education and Research. 2019;7(11): 57– 70.
16. Mohammed Muniru Iddrisu, Kodjoe Isaac Tetteh. The Gamma Function and Its Analytical Applications". Journal of Advances in Mathematics and Computer Science. 2017;23(3):1-16.
17. Yingpu Deng and Yanbin Pan. An Algorithm for Factoring Integers. CiteSeerX; 2018.
18. Peter L. Montgomery. A Survey of Modern Integer Factorization Algorithms. CWI Quarterly. 1994;7(4):337 – 365.
19. David CW Muir. Factoring Integers using Geometry; 2022.
20. Seema Kute, Chitra Desai, Mukti Jadhav. Analysis of factoring algorithms for number factorization. International Journal of Creative Research Thoughts. 2023;11(3): 735 – 740.
21. Abdelmajid Hassan Mansour. Analysis of RSA Digital Signature Key Generation using Strong Prime. International Journal of Computer (IJC). 2017;24(1):28-36.
22. Venkateswara Rao Pallipamu, Thammi Reddy K, Suresh Varma P. Design of RSA Digital Signature Scheme Using A Novel Cryptographic Hash Algorithm. International Journal of Emerging Technology and Advanced Engineering. June 2014;4(6).
23. Farah Jihan Aufa; Endroyono, Achmad Affandi. Security System Analysis in Combination Method: RSA Encryption and Digital Signature Algorithm. 4th International Conference on Science and Technology (ICST); 2018.
24. Om Pal and Bashir Alam. Diffie-Hellman Key Exchange Protocol with Entities

Authentication. International Journal of Engineering and Computer Science. April 2017;6(4):20831-20839

25. Chiradeep Gupta, Subba Reddy NV. Enhancement of security of Diffie-hellman key exchange protocol using RSA Cryptography. Journal of Physics: Conference Series; 2021.
26. Shahid Mubeen, Abdur Rehman. (n, k)-FACTORIALS. Journal of Inequalities and Special Functions 2014;5(3):14-20.
URL: http://www.ilirias.com
ISSN: 2217-4303
27. Babu G. Properties of Combination Using Double Factorial. International Journal of Science and Research (IJSR); 2016.
28. Bo Xiong, Yimin Huang, Steffen Staab, Zhenguo Li. MOFA: Modular Factorial Design for Hyper-Parameter Optimization; 2020;
Available:https://www.arxiv.org
29. Man-Hong Yung. Quantum supremacy: Some fundamental concepts. Oxford University Press; 2017.
30. Neetu Settia. Cryptanalysis of modern cryptographic algorithms. International Journal of Computer Science and Technology. 2010;1(2):166 – 169.
31. Ibrahim Marouf, Mohammed Mosab Asad, Qasem Abu Al-Haija. Comparative Study of Efficient Modular Exponentiation Algorithms. COMPUSOFT. An International Journal of Advanced Computer Technology. August 2017;6(8): 2381– 2389.
32. Anthony Overmars, Sitalakshmi Venkatraman. A New Method for Factorizing Semi-Primes Using Simple Polynomials. 3rd International Conference on Research in Applied Science. 2020;25 – 30.
33. Anthony Overmars, Sitalakshmi Venkatraman. A fast factorisation of semi-primes using sum of squares. Mathematical and Computational Application; 2019.
34. Anthony Overmars, Sitalakshmi Venkatraman. New Semi-Prime Factorization and Application in Large RSA Key Attacks. Journal of Cybersecurity and Privacy; 2021.
35. A Fast Factorization of Semi Primes Using Sum of Squares. 2019;11-12.
36. Adeniyi Abidemi Emmanuel, Okeyinka Aderemi E, Adebiyi Marion O, Asani Emmanuel O. A Note on Time and Space Complexity of RSA and El Gamal Cryptographic Algorithms. IJACSA) International Journal of Advanced Computer Science and Applications. 2021;12(7).
37. Keita Emura, Atsuko Miyaji, Kazumasa Omote, Akito Nomura, Masakazu Soshi. A Ciphertext-Policy Attribute-Based Encryption Scheme with Constant Ciphertext Length. Conference Paper in International Journal of Applied Cryptography· November 2009. Japan Advanced Institute of Science and Technology; 2009.
38. Zhengping Jay Luo, Ruowen Liu, Aarav Mehta. Understanding the RSA algorithm". ACM Vol. 1, No. 1, Article. Publication date: August 2023; 2023.
39. Robert Erra, Christophe Grenier. The Fermat factorization method revisited; 2009.
Available:https://eprint.iacr.org/2009/318.pdf

© Copyright (2024): Author(s). The licensee is the journal publisher. This is an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Peer-review history:
The peer review history for this paper can be accessed here:
https://www.sdiarticle5.com/review-history/115573