CDMI: A Clockwise-Displacement Algorithm to Compute Multiplicative Inverse

Hashim Abu-gellban
Department of Computer Science
Texas Tech University, USA
hashim.gellban@ttu.edu

Long Nguyen
Whitacre College of Engineering
Texas Tech University, USA
long.nguyen@ttu.edu

***Abstract*—A multiplicative inverse (MI) algorithm is used in several fields, like cryptography algorithms. There are many MIs, however, these algorithms suffer from using several multiplication and division operations, which take more execution time than additions and subtractions. We created a new algorithm called Clockwise-Displacement (CDMI) by using addition and subtraction operations in the iterative steps instead of multiplications and divisions. Additionally, numerous MIs face the undecidable problem because of the floating-point issue. Whereas, CDMI tackles this issue by converting the domain from the floating-point space to integer space. Therefore, CDMI declines the time consuming to calculate the multiplicative inverse by applying fewer divisions and multiplications (expensive operations) and addresses the rounding error issue in some MI Algorithms.**

***Index Terms*—Multiplicative Inverse, Security, Public Key Techniques, Asymmetric Techniques, Cryptography.**

I. Introduction

Many cryptography algorithms depend on computing the multiplicative inverse [1], [2], [3]. Therefore, it is crucial to compute the inverse in less time to increase the performance of these cryptography algorithms and to lower the CPU usage. Rivest et al. [1], [4], [5] used multiplicative inverse positive integer number (d) to compute the encryption key (e, n) and the decryption key (d, n). The encryption key and the decryption key are called the public key and the private key, respectively. $e.d \equiv 1 \bmod (1-p)(1-q)$, where p, q are prime numbers and e is a positive integer ($1 < e < n$) which is a part of the encryption key. $n = p.q$ and $\phi(n) = (1-p)(1-q)$. We can rewrite the formula: $e.d \equiv 1 \bmod \phi(n)$. $\phi(n)$ is a positive number and is not a prime number. Also, the variables e and $\phi(n)$ are relatively primes, which means that there is no common divisor number between them except 1 (i.e., $\gcd(e, \phi(n)) = 1$). We can encrypt a message (M) to get a ciphertext (C) using the following formula: $C = M^e \bmod n$. To decrypt the ciphertext (C) to get the original message (M), we can use $M = C^d \bmod n$. Moreover, security is ubiquitous and MI can be applied in many areas, such as Cyber-Physical Systems (CPS) [6], web applications [7], [8], [9], [10], or other domains [11], [12].

Gordon algorithm used two loops with shift operations to avoid the multiplication and division operations in the Extended-Euclidean Algorithm [13]; however, it takes several steps to find the solutions [14]. The Fast Fraction algorithm suffers from fraction issue by dividing two real numbers inside iterative steps until we get an integer number [15].

The purpose of this paper is to find the multiplicative inverse integer number (d) in the formula: $e.d \equiv 1 \bmod \phi(n)$ in a fast way by reducing the computational time of the exhausted division and multiplication operations. Our new algorithm (Clockwise-Displacement) uses only additions and subtractions in the iterative steps. Additionally, it only uses the multiplications and the divisions in the first and last steps, which are not a part of the loop. More particularly, our main contributions in this paper are:

- To the best of our knowledge, we are one of the first in using addition and subtraction operations instead of multiplications and divisions in the iterative parts to improve the time complexity to compute the multiplicative inverse.
- We address the fraction issue (i.e., the rounding error) of the Baghdad algorithm by using modulo operation, having only integer variables in our proposed algorithm.

II. Baseline Methods

In this Section, we describe 3 comparative algorithms by briefing the advantages and disadvantages of these methods.

A. Extended-Euclidean Algorithm

The general idea of this method is to find d that is the multiplicative inverse of e where d and $\phi(n)$ satisfy $e.d \equiv 1 \bmod (1-p)(1-q)$. The power of this method is applying a fast way to search deep, to find the solution as every iteration divides g by u. The main drawback of the algorithm is that it uses a large number of variables (i.e., g, i, v, u, q, and t) to find the solution. The algorithm is described in Algorithm 1, and an example of the algorithm is shown in Table I. The multiplicative inverse positive integer number (d) is 611 (i.e., $d = i + \phi(n) = -229 + 840 = 611$).

B. Baghdad Algorithm

This method [14] depends on the formula $e.d \equiv 1 \bmod \phi(n)$ which means $e.d = 1 + k.\phi(n)$ for a positive integer number k [15]; so, $d = \frac{1+k.\phi(n)}{e}$ where d is a positive integer number. The algorithm is presented in Algorithm 2. The pros of this algorithm are: (1) it has a small number of variables; (2) it is not a complex algorithm. However, Aboud [14] has mentioned

Algorithm 1: Extended-Euclidean Algorithm.

function Extended_Euclidean $(e, \phi(n))$
Input : $\gcd(e, \phi(n)) = 1$, where $e \in \mathbb{Z}_{\phi(n)}$
Output: $e^{-1} \bmod \phi(n)$, where $e^{-1} = d$ provided that d exists
1. let $g \leftarrow \phi(n), i \leftarrow 0, v \leftarrow 1$, and $u \leftarrow e$
2. While $(u > 0)$ perform:
$q \leftarrow \lfloor \frac{g}{u} \rfloor$, $t \leftarrow g - q * u$, $g \leftarrow u$, $u \leftarrow t$
$t \leftarrow i - q * v$, $i \leftarrow v$, $v \leftarrow t$
3. if $(i < 0)$ then $i \leftarrow i + \phi(n)$
4. compute: $d \leftarrow i$
return d

TABLE I: An Example of Extended-Euclidean Algorithm. $Given : \phi(n) \leftarrow 840, e \leftarrow 11$. $The\ output : d \leftarrow 611$.

Step	g	i	v	u	q	t
init	840	0	1	11	0	0
1	11	1	-76	4	76	-76
2	4	-76	153	3	2	153
3	3	153	**-229**	1	1	-229
4	1	-229	**840**	0	3	840

that the method suffers from large values of e because the calculation of d adds accumulative large fraction numbers $(1/e)$ in every iterative step. As a consequence of the fraction issue, the algorithm may not return the multiplicative inverse (d); in this case, the algorithm faces the infinite loop issue, since the integer condition of the loop is unreachable. In the Extended-Euclidean algorithm, there are no fraction decimals that make the method scalable for large values of e; therefore, it returns a correct value of d if exists.

Algorithm 2: Baghdad Algorithm.

function baghdad $(e, \phi(n))$
Input : $\gcd(e, \phi(n)) = 1$, where $e \in \mathbb{Z}_{\phi(n)}$
Output: $e^{-1} \bmod \phi(n)$, where $e^{-1} = d$ provided that d exists
1. $d \leftarrow \frac{1}{e}$
2. Repeat:
$d \leftarrow d + \frac{\phi(n)}{e}$
Until: d is integer
return d

Table II shows an example of the algorithm where there is no output because the algorithm goes in an endless loop without satisfying the condition (d is an integer number) to break the loop. This algorithm is $undecidable$ [16] even when there is a solution for given inputs. In step 8, the algorithm should have found the solution; however, d is still not an integer number with a fraction (0.000061).

C. Fast Fraction Algorithm

This algorithm [15] depends on the formula $e.d \equiv 1 \bmod \phi(n)$ as the Baghdad algorithm [14]. The Fast Fraction algorithm uses several division operations to quickly find the multiplicative inverse. The key advantage of this algorithm is that it needs fewer steps to find d. However, this method also suffers from large values of e since it gives a large number of decimal points (fraction issue) for the variable r. In this case, the fast fraction algorithm goes in an endless loop without returning back the result (i.e., $undecidable$ problem). The details are explained in Algorithm 3.

TABLE II: An example of the Baghdad Algorithm. $Given : \phi(n) \leftarrow 840, e \leftarrow 11$. $There\ is\ no\ output$.

Step	d	The result of the condition
1	76.454552	d is not integer
2	152.818192	d is not integer
3	229.181824	d is not integer
4	305.545471	d is not integer
5	381.909119	d is not integer
6	458.272766	d is not integer
7	534.636414	d is not integer
8	**611.000061**	**d is not integer**
9	687.363708	d is not integer
10	763.727356	d is not integer
11	840.091003	d is not integer
...	...	d is not integer

Algorithm 3: Fast Fraction Algorithm

function fast_fraction $(e, \phi(n))$
Input : $\gcd(e, \phi(n)) = 1$, where $e \in \mathbb{Z}_{\phi(n)}$
Output: $e^{-1} \bmod \phi(n)$, where $e^{-1} = d$ provided that d exists
1. $sf \leftarrow \frac{(\phi(n)+1) \bmod e}{e}$, $df \leftarrow \frac{\phi(n) \bmod e}{e}$, $i \leftarrow 1$
4. if $(sf = 0)$, then return no solution
5. Repeat:
$r \leftarrow \frac{i - sf}{df}$, $i \leftarrow i + 1$
Until: r is integer
6. $d \leftarrow \frac{1 + \phi(n).(r+1)}{e}$
return d

Table III contains an example of the Fast Fraction Algorithm. From the last step $r = 7$, the multiplicative inverse integer number (d) is 611 (i.e., $d = \frac{1+840*(7+1)}{11} = 611$). The algorithm finishes the loop quickly in 3 steps.

TABLE III: An example of the Fast Fraction Algorithm. $Given : \phi(n) \leftarrow 840, e \leftarrow 11$. $The\ output : d \leftarrow 611$.

Step	sf	df	r	The result of the condition
1	0.454545	0.363636	1.5	r is not integer
2	0.454545	0.363636	4.25	r is not integer
3	0.454545	0.363636	7	r is integer

III. Methodology

A. Clockwise-Displacement Algorithm

Our method addresses two issues. First, CDMI solves the issue of adding numerous fractions in the iterative process that we have described in the Baghdad and the Fast Fraction Sections. Second, it is faster than these two algorithms by using only additions and subtractions in the iterative steps instead of division and multiplication operations. The division and multiplication operations are expensive. One of the fastest multiplication algorithm [17], [18] is called Fürer which takes

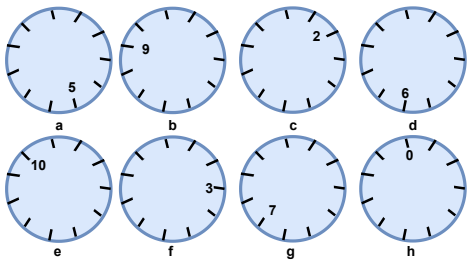

Fig. 1: A demo of how modular arithmetic works like a clock in the Clockwise-Displacement Algorithm starting from step (a) ending with step (h). Our clock has 11 ticks ($e = 11$).

$O(n \log n \,.\, 2^{O(\log n)})$ for multiplying two n-integer numbers, comparing to addition operations with time complexity $\theta(n)$. Because of this, our method is faster than the other algorithms for small numbers of input e except for the Extended-Euclidean algorithm using some inputs. However, our new algorithm may take several iterations to find the ideal i that is important to compute d for some large values of e. In this case, it is slower than the Extended-Euclidean method. Algorithm 4 describes the steps in more detail.

Algorithm 4: Clockwise-Displacement Algorithm.

function clockwise_displacement $(e, \phi(n))$
Input : $\gcd(e, \phi(n)) = 1$, where $e \in \mathbb{Z}_{\phi(n)}$
Output: $e^{-1} \bmod \phi(n)$, where $e^{-1} = d$
provided that d exists
1. $\Delta \leftarrow \phi(n) \bmod e, i \leftarrow 0, m \leftarrow 1$
2. Repeat:
$m \leftarrow m + \Delta$, if $m \geq e$ then $m \leftarrow m - e$
$i \leftarrow i + 1$
Until: $m = 0$
3. $d \leftarrow \frac{1+i.\phi(n)}{e}$
return d

B. An Example of the Clockwise-Displacement Algorithm

This algorithm finds the solution in 8 iterations as it is presented in Table IV. In the last step, the value of i is equal to 8, where $m = 0$ gives $d = \frac{i.\phi(n)+1}{e}$ which means d is an integer number. So, $d = \frac{8*840+1}{11} = \frac{6720+1}{11} = 611$. Even it takes 8 iterations as the Baghdad algorithm should have done, but it finishes faster as it is shown in Section IV since it uses mainly the addition and subtraction operations in the iterative part of this algorithm.

Fig. 1 illustrates how the displacement Δ with modulus operator makes finding the solution as a clock of 11 numbers simpler and as it moves 4 jumps ($\Delta = 4$) for each step in a clockwise rotation. $m = 5$ in the first iteration (a). After jumping according ($\Delta = 4$), $m = 9$. After the 8^{th} iteration, $m = 0$, that reaches the end of the loop.

C. Proof of the Clockwise-Displacement Algorithm

It is known that $(a + b) \bmod x = (a \bmod x + b \bmod x) \bmod x$. Also, $(a.b) \bmod x = (a \bmod x \,.\, b \bmod x) \bmod x$.

TABLE IV: An example of the Clockwise-Displacement Algorithm. *Given* : $\phi(n) \leftarrow 840, e \leftarrow 11$. *The output* : $d \leftarrow 611$.

Step	Δ	m
1	4	5
2	4	9
3	4	2
4	4	6
5	4	10
6	4	3
7	4	7
8	4	0

First, we need to show that the multiplicative inverse is $d = \frac{1+i.\phi(n)}{e}$ when m is zero. In the iteration part (step 2) of Algorithm 4, we compute $m \leftarrow (m + \Delta) \bmod e$ that is equivalent to the addition statement $m \leftarrow m + \Delta$ and the if statement (if $m \geq e$ then $m \leftarrow m - e$) since $\Delta \leftarrow \phi(n) \bmod e$ (step 1) gives $\Delta \in \mathbb{Z}_e$. Therefore, the addition and the if statements are the simplified form of modulus (e). After the i^{th} iteration, it can be represented by $m \leftarrow (1 + i.\Delta) \bmod e$ gives $m \leftarrow (1 + i.(\phi(n) \bmod e)) \bmod e$ gives $m \leftarrow (1 + i.\phi(n)) \bmod e$. So, $1 + i.\phi(n)$ is the multiples of e since $(m = 0)$. At step 4, $d \leftarrow \frac{1+i.\phi(n)}{e}$ is an integer number. Thus, d is the multiplicative inverse, since $e.d \equiv 1 \bmod \phi(n)$ gives $d = \frac{1+i.\phi(n)}{e}$ [14].

Second, we prove our algorithm by induction. If d is the multiplicative inverse of e, then:

- $e.d \equiv 1 \bmod \phi(n)$
- $d \leftarrow \frac{1+C.\phi(n)}{e}$ where C is a constant positive integer number [14].
- $\gcd(e, \phi(n)) = 1$.
- $e, d \in \mathbb{Z}_{\phi(n)}$, where $\phi(n)$ is a positive integer number.

Base case: the first iteration ($i = 1$ and $m = 0$), we have: $d = \frac{1+1.\phi(n)}{e} = \frac{1+\phi(n)}{e} \rightarrow e.d = 1 + \phi(n)$ true

Inductive step: assume it is true for k iterations and $m = 0$: $d = \frac{1+k.\phi(n)}{e} \rightarrow e.d = 1 + k.\phi(n)$ true

Then, for the $(k + 1)^{th}$ iteration ($i = k + 1$ and $m = 0$): $d = \frac{1+(k+1).\phi(n)}{e} \rightarrow e.d = 1 + (k + 1).\phi(n)$ true, since $k + 1$ is a constant and a positive integer number.

IV. Experiments and Results

A. Experiments

We used in our experiments a sample of inputs as shown in Table V. d values are between 7 and 907 and $\phi(n)$ values are less than or equal 8 digits (between 34200 and 98962380). We ran each algorithm on the dataset 10 times and computed the execution time. We removed the maximum and minimum execution time from our results and computed the average of the rest running times.

TABLE V: A sample of inputs

e	$\phi(n)$	e	$\phi(n)$	e	$\phi(n)$
7	18648036	97	50172	191	8795304
11	393520	107	18648036	197	20007108
23	98962380	109	18648036	907	18648036
89	34200	131	18648036		

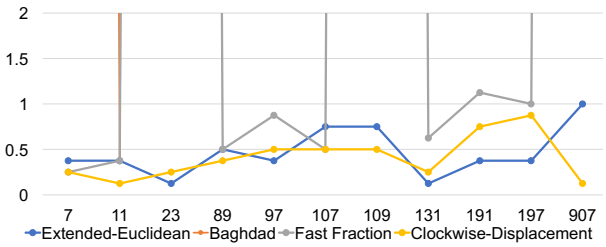

Fig. 2: Average Execution Time ("μs").

TABLE VI: Total time to run the algorithms. The bold font shows the best result.

Algorithm	**Total Average Time (μs)**
Extended-Euclidean	5.125
Baghdad	$>$ 100.000
Fast Fraction Algorithm	$>$ 100.000
CDMI	**4.500**

B. Results

We ran our experiments using different input values (e and $\phi(n)$) as shown in Table V. The x-axis in Fig. 2 represents the values of e while the y-axis shows the average execution time ("μs") for all algorithms. CDMI found the multiplicative inverse in less than (1μs) for different values of e. The Extended-Euclidean method was performed similarly to CDMI. Whereas, our algorithm outperformed all baseline algorithms in the total running time for all inputs as shown in Fig. 2. This improvement is because we accelerate the execution time of the algorithm by depending more on addition and subtraction arithmetic operations and only use the modulus, multiplications, and divisions in the first and the last steps. Additions and subtractions have less time complexity than multiplications and divisions. In other words, no multiplication and division operations are used in the iterative steps while other algorithms applied heavily to the multiplication and division operations in their loops. Table VI shows that our algorithm took the least total average time (4.500μs). The Baghdad algorithm went in an endless loop for all examples except for the second inputs ($e = 11, \phi(n) = 393520$), while the Fast Fraction algorithm faced the endless loop for several inputs (e.g., $e = 23$). Our algorithm and the Euclidean algorithm found the solution in a reasonable time without any fraction issue. However, the other algorithms got stuck in infinite loops. This is because the Clockwise-Displacement and the Euclidean algorithm do not use real variables (i.e., no fractions) to find the multiplicative inverse d.

V. Conclusion

Finding multiplicative inverse can be calculated by different methods. We compare our algorithm (Clockwise-Displacement) with the based methods to compute the multiplicative inverse. Our algorithm performs similar to the Extended-Euclidean algorithm in the experiments, and CDMI outperforms other baseline methods. This is because our algorithm uses the addition and subtraction operations (less expensive than the multiplication and division operations) in the iterative steps without any fraction to find the multiplicative inverse. Using addition and subtraction operations addresses the Baghdad and Fast Fraction methods issue (the infinite loop issue) and improve the performance. Additionally, our algorithm aims to address the fraction issue by converting the domain from the floating-point space into integer space. Furthermore, cryptography recommends using small values of e for security reasons [1], [19], [14]. For future work, we will study the Fast Fraction and other algorithms to improve their performance and address their fraction issues.

References

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A methodfor obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[2] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, 2018.

[3] S. Blazy and R. Hutin, "Formal verification of a program obfuscation based on mixed boolean-arithmetic expressions," in *8th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM, 2019, pp. 196–208.

[4] X. Lai and J. L. Massey, "A proposal for a new block encryption standard," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1990.

[5] S. J. Aboud, M. A. AL-Fayoumi, M. Al-Fayoumi, and H. S. Jabbar, "An efficient rsa public key encryption scheme," in *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference*. IEEE, 2008.

[6] H. Abu-gellban, L. Nguyen, M. Moghadasi, Z. Pan, and F. Jin, "Livedi: An anti-theft model based on driving behavior," in *Proceedings of the 2020 ACM Workshop on Information Hiding and Multimedia Security*, 2020, pp. 67–72.

[7] L. H. Nguyen, S. Jiang, H. Abu-gellban, H. Du, and F. Jin, "Nipred: Need predictor for hurricane disaster relief," in *Proceedings of the 16th International Symposium on Spatial and Temporal Databases*, 2019, pp. 190–193.

[8] H. Du, L. Nguyen, Z. Yang, H. Abu-gellban, X. Zhou, W. Xing, G. Cao, and F. Jin, "Twitter vs news: Concern analysis of the 2018 california wildfire event," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2019, pp. 207–212.

[9] H. Abu-gellban, "A survey of real-time social-based traffic detection," *arXiv preprint arXiv:2007.04100*, 2020.

[10] O. Adwan, M. Al-Tawil, A. Huneiti, R. Shahin, A. A. Zayed, and R. Al-Dibsi, "Twitter sentiment analysis approaches: A survey," *International Journal of Emerging Technologies in Learning (iJET)*, vol. 15, no. 15, pp. 79–93, 2020.

[11] M. Al-Tawil, V. Dimitrova, and D. Thakker, "Using knowledge anchors to facilitate user exploration of data graphs," *Semantic Web*, no. Preprint, pp. 1–30, 2020.

[12] M. Al-Tawil, V. Dimitrova, D. Thakker, and A. Poulovassilis, "Evaluating knowledge anchors in data graphs against basic level objects," in *International Conference on Web Engineering*. Springer, 2017, pp. 3–22.

[13] J. Gordon, "Fast multiplicative inverse in modular arithmetic," *cryptography and coding*, pp. 269–279, 1989.

[14] S. J. Aboud, "Baghdad method for calculating multiplicative inverse," *Proceedings. (ITCC) 2004. International Conference*, vol. 2, 2004.

[15] H. M. Al-Matari, S. J. Aboud, and N. F. Shilbayeh, "Fast fraction-integer method for computing multiplicative inverse," in *arXiv preprint*. arXiv, 2009.

[16] M. Bojańczyk, E. Kelmendi, and M. Skrzypczak, "Mso + delta is undecidable," in *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2019, pp. 1–13.

[17] M. Fürer, "Faster integer multiplication," *SIAM Journal on Computing*, vol. 39, no. 3, 2009.

[18] D. J. Bernstein, "Fast multiplication and its applications," *Algorithmic number theory*, vol. 44, 2008.

[19] D. Boneh, "Twenty years of attacks on the rsa cryptosystem," in *Notices of the American Mathematical Society (AMS)*, vol. 46, no. 2. AMS, 1999, pp. 203–213.