Int. J. Math. And Appl., 9(3)(2021), 61–66

ISSN: 2347-1557

Available Online: http://ijmaa.in/

International Journal of Mathematics And its Applications

Applications of Modular Arithmetic and Recursion to the RSA Cipher

Eshaan Giri[1,*]

1 Millard North High School, Omaha, Nebraska, United States.

Abstract: Regarding cryptography, one of the most popular forms of encoding information today is the RSA Cipher, used to convert one integer into another. The workings of the RSA Cipher can be described through the lens of modular arithmetic, a branch of number theory focused on remainders when dividing positive integers. This paper will explain the mathematics of the RSA Cipher, methods of decryption, and justifications as to why it is such an effective tool for encoding data.

MSC: 11A05.

Keywords: Modular Arithmetic, Euclidean Algorithm, Recursion, RSA Cipher.

1. Introduction

In today's world, keeping certain information private is an essential facet of life, whether the information in question is a social security number, a simple embarrassing moment back in middle school, or something else entirely. However, this information sometimes has to be shared with another trusted individual. Dumbledore's Army had to broadcast meeting dates without Umbridge knowing, Eeth Koth had to inform Obi-Wan Kenobi of his location without alerting General Grievous, and I had to turn in a late assignment without losing face in front of my classmates. In all of these examples, messages had to be concealed to all but the intended receiver. The encoding of information in such a fashion is called a cipher. Ciphers can come in many forms; in Harry Potter, serial numbers on fake currency were used; in Star Wars: The Clone Wars, hand signals were implemented; in my case, I just used email. Regardless of the particular method, every single one of these examples involved transmitting information indecipherable by an outside observer to an intended receiver. Any such encoding of information is called a cipher. In this paper, we will discuss one of the most popular ciphers in use today, the RSA Cipher [1, 2].

2. The RSA Cipher

The RSA Cipher is a popular method of encoding numerical data by means of modular arithmetic. In essence, it converts one long integer, which could represent a message in any form, into another long integer, which can then be decoded by a receiving party into the original message. In order to convert the original integer, i.e. 'plaintext', into an encoded 'ciphertext',

* *E-mail: eshaan.giri@gmail.com (Student)*

the following process is used: Given two large primes p and q, an integer k relatively prime to both $p-1$ and $q-1$, and plaintext x, the resulting ciphertext b can be represented by

$$b \equiv x^k \pmod{m}, \tag{1}$$

where $m = pq$. Given the basic formula for encryption, the question of which variables are known by particular parties arises naturally. As will be proven later in 3.1, given that p and q are sufficiently large, one cannot decode ciphertext without knowledge of p and q, regardless of whether or not m or k are known. Thus, the latter two variables can be broadcast publicly, while the former two must be kept private, known only to the sender and intended receiver.

3. Mathematics of Decryption

3.1. Outline

Assuming that p, q, and k are known and that one has received ciphertext b, the method of decryption is outlined below. The various aspects of the process as well as the reasoning between the steps below will be described in detail in the following sections.

(1). Find $\phi(m)$ by applying Euler's totient function.

(2). Derive a solution $u, v \in \mathbb{Z}^+$ to the equation $ku = 1 + v\phi(m)$.

(3). Compute $x \equiv b^u \pmod{m}$.

3.2. Euler's Totient Function

Leonhard Euler defined the function $\phi(k)$ as the number of positive integers less than k relatively prime to k. $\phi(k)$ can be computed by the expression $k\prod_{p|k}\frac{p-1}{p}$, where p represents any prime number that is a factor of k.
A general proof of this formula could be stated, but for the purposes of RSA Encryption, the only totient necessary to compute is that of $m = pq$ for $p, q \in \mathbb{P}$, in which case the formula above collapses to $\phi(m) = (p-1)(q-1)$.
This formula can be corroborated by complementary counting on the set of all numbers less than or equal to m. In order to calculate the amount of numbers relatively prime to m, we can subtract the amount of multiples of each p and q from m, as these are the only numbers that would share a factor with m. As $m = pq$, there would be q multiples of p and p multiples of q less than or equal to m. Subtracting yields $m - q - p$. However, the issue of overcounting must be addressed, as the quantity m is a multiple of both p and q. It was thus subtracted twice, and the appropriate correction can be made, giving $\phi(m) = m - p - q + 1$, which of course equals $pq - p - q + 1 = (p-1)(q-1)$.

3.3. Euler's Totient Theorem

This function becomes particularly useful due to Euler's additional proof that, for any integer c such that $\gcd(c, k) = 1$, $c^{\phi(k)} \equiv 1 \pmod{k}$. Proof of this theorem is shown below.

Theorem 3.1. *For any integer c such that $\gcd(c,k) = 1$, $c^{\phi(k)} \equiv 1 \pmod{k}$.*

Proof. Let $A = \{a_1, a_2, ..., a_{\phi(k)}\} \pmod{k}$ such that A is the set of all integers less than k relatively prime to k. We now postulate that A is congruent to $B = \{c(a) : a \in A\} \pmod{k}, c \in \mathbb{Z}^+, \gcd(c,k) = 1$. As all the elements of A are distinct by definition of A, multiplying each element of the set by a constant relatively prime to k will yield another set with distinct elements. As each element of B shares no factors with k, it must be true that the elements of A are the elements of B.

Given that these two sets are equivalent $\pmod{k}$, it then follows that $\prod_{i=1}^{\phi(k)} a_i \equiv \prod_{i=1}^{\phi(k)} b_i \pmod{k}$, with a_i and b_i being the elements of A and B respectively, which expands to $\prod_{i=1}^{\phi(k)} a_i \equiv c^{\phi(k)} \prod_{i=1}^{\phi(k)} a_i \pmod{k}$. The products on either side of the equation cancel, leaving $c^{\phi(k)} \equiv 1 \pmod{k}$. $\square$

3.4. Diophantine Equation

In order to apply the theorem above to the decryption of an RSA cipher, a solution $u, v \in \mathbb{Z}^+$ to the equation $ku = 1 + v\phi(m)$ must be found. Once found, Euler's totient theorem trivializes the decryption process, as will be shown in the next section. It is at this point that a computerised recursion algorithm can be implemented to great effect.

To begin, we can re-write our equation into a form that will allow us to generalise the problem. We can restate our equation as $u(k) + v(-\phi(m)) = \gcd(k, -\phi(m))$. We can see that $\gcd(k, -\phi(m)) = 1$ because of the condition outlined in Section 2 that k is relatively prime to both $p-1$ and $q-1$. If k is relatively prime to all factors of $\phi(m)$, it must be relatively prime to $\phi(m)$ itself. Note that the sign of $\phi(m)$ has no bearing on this outcome. We can now approach the more general problem of calculating what are known as "Bézout Coefficients", the term for which derives from Bézout's Identity, stated below.

Identity 3.2 (Bézout's Identity)**.** *For $a, b \in \mathbb{Z}^+$ with a greatest common divisor of d, there always exist coefficients u and v such that $ua + vb = d$.*

For proof, we can turn to one of the most popular and efficient methods of calculating the greatest common divisor of two integers; the Euclidean Algorithm. As it turns out, back substituting values obtained in each step of this algorithm allows us to calculate Bézout Coefficients quite nicely. The Euclidean Algorithm chiefly relies on the trivial fact that $\gcd(a, b) = \gcd(a, b \pmod{a})$. After all, if $b = ca + r$, any divisor of a will divide ca, so any common factor between a and b must divide r.

To find Bézout Coefficients, we can employ the Extended Euclidean Algorithm in two phases; the first phase, the "forward" substitution, will calculate $\gcd(a, b)$; the second phase, the "backward" substitution, will express $\gcd(a, b)$ in terms of intermediate values found in the first phase.

The Extended Euclidean Algorithm is as follows for two integers a, b with $a \leq b$:

Phase 1: Forward

(1). Generate a list $\{a_n, b_n\}$ with $(a_0, b_0) = (a, b)$ such that $(a_{i+1}, b_{i+1}) = (b_i, a_i \pmod{b})_i)$.

(2). Stop once $b_k = 0$. $a_k = \gcd(a_0, b_0)$.

Phase 2: Backward

(1). Generate a list $\{u_n, v_n\}$ with $(u_k, v_k) = (1, 0)$ such that $(u_{j-1}, v_{j-1}) = \left(v_j, u_j - \left\lfloor \frac{a_{j-1}}{b_{j-1}} \right\rfloor v_j\right)$.

(2). Stop once (u_0, v_0) have been found. These values will satisfy Bézout's identity for a and b.

It can be easy to get lost in the mess of variables above, so an example is shown below.

Example 3.3. *We are asked to find $\gcd(35, 63)$. We can start Phase 1 with $(a_0, b_0) = (35, 63)$. Values of a_n and b_n are shown in the table below.*

Index (i)	a_i	b_i
0	35	63
1	63	35
2	35	28
3	28	7
4	7	0

We thus know that $7 = \gcd(35, 63)$. *Now, we can start Phase 2 with* $(u_4, v_4) = (1, 0)$. *Values of* u_n *and* v_n *are shown in the table below. Note that the indices are shown in decreasing order to reflect the nature of back-substitution.*

Index (i)	u_i	v_i
4	1	0
3	0	1
2	1	-1
1	-1	2
0	2	-1

Given $(u_0, v_0) = (2, -1)$, *we can successfully verify that* $35(2) + 63(-1) = 7$.

For justification that this method will succeed for any (a, b), we can turn to mathematical induction. Our base case would be the index n such that $b_n = 0$, as it is trivial that $a_n = 1(a_n) + 0$. The inductive step is proving that, for any index k, $a_k u_k + b_k v_k = \gcd(a_k, b_k)$ implies that $a_{k-1}u_{k-1} + b_{k-1}v_{k-1} = \gcd(a_{k-1}, b_{k-1})$.
Firstly, we can see that $\gcd(a_k, b_k) = \gcd(a_{k-1}, b_{k-1})$, so our problem simplifies to proving that

$$a_k u_k + b_k v_k = a_{k-1}u_{k-1} + b_{k-1}v_{k-1}$$

By the nature of the Euclidean transformations amongst $\{a_n\}$ and $\{b_n\}$, we know that $a_k = b_{k-1}$ and that $b_k = a_{k-1} \pmod{b_{k-1}} = a_{k-1} - \left\lfloor \frac{a_{k-1}}{b_{k-1}} \right\rfloor b_{k-1}$. By the nature of the transformations amongst $\{u_n\}$ and $\{v_n\}$, we know that $u_{k-1} = v_k$ and that $v_{k-1} = u_k - \left\lfloor \frac{a_{k-1}}{b_{k-1}} \right\rfloor v_k$. Thus, we can substitute these values into the original equation, yielding the repeated simplifications shown below.

$$\begin{aligned}
a_k u_k + b_k v_k &= a_{k-1}v_k + a_k v_{k-1} \\
a_k u_k + (a_{k-1} - \left\lfloor \frac{a_{k-1}}{b_{k-1}} \right\rfloor b_{k-1})v_k &= a_{k-1}v_k + a_k v_{k-1} \\
a_k u_k + a_{k-1}v_k - \left\lfloor \frac{a_{k-1}}{b_{k-1}} \right\rfloor b_{k-1}v_k &= a_{k-1}v_k + a_k v_{k-1} \\
a_k u_k - \left\lfloor \frac{a_{k-1}}{b_{k-1}} \right\rfloor a_k v_k &= a_k v_{k-1} \\
a_k(u_k - \left\lfloor \frac{a_{k-1}}{b_{k-1}} \right\rfloor v_k) &= a_k v_{k-1} \\
a_k v_{k-1} &= a_k v_{k-1}.
\end{aligned}$$

As all of the substitutions used can be reversed, we have justified that our generation does indeed yield the Bézout coefficients of any two integers.
To write this in code, around only five lines of code are needed, as this is a relatively elegant recursion. For example, code in python is shown below.

```
def BCoeffs(a,b):

    if b == 0:
        return (a,1,0)
    else:
        g,u,v = BCoeffs(b,a%b)
        return (g,v,u-(a//b)*v)
```

Note that in python, $h\%k$ signifies $h \pmod{k}$ and that $h//k$ signifies $\lfloor \frac{h}{k} \rfloor$.

3.5. Final Steps

Armed with the solution above, the final step in decryption seems almost effortless. Remembering the formula for the original ciphertext

$$b \equiv x^k \pmod{m} \tag{1 revisited}$$

we can simply raise both sides of this equivalence to the power of u, yielding $b^u \equiv x^{ku} \pmod{m}$. The quantity ku can now be substituted with $1 + v\phi(m)$, yielding $b^u \equiv x^{1+v\phi(m)} \pmod{m}$ By Euler's Totient Theorem, $x^{v\phi(m)} \equiv 1 \pmod{m}$, so the equivalence simplifies to $b^u \equiv x \pmod{m}$. As our original goal was to compute the original plaintext x, we have arrived at our solution.

4. Specifics of Use

4.1. Public and Private Keys, Revisited

The specific variables needed to be kept private and public have been outlined in 1.3. Now, a justification will be given as to why p and q are the only variables needed to be kept private, given that they are sufficiently large. We can approach this problem from the point of view of one attempting to decode the RSA ciphertext b without p and q, given that we know m and k. As outlined in the previous section, we first need to find $\phi(m)$. To do this, we need to find p and q by factoring m. Unfortunately, the fastest classical algorithm for such a computation, the General Number Field Sieve, takes months to factor a number with as small as 140 digits. In practical application of the RSA Cipher, m could be set far larger by generating larger p and q, which can be done quite easily with a variety of algorithms. In essence, because it is far easier to generate a product of large primes than it is to factor such a product, p and q cannot be efficiently determined even if m is known. If these two quantities cannot be determined, the RSA Cipher cannot be deciphered.

4.2. Translating Text as an Integer

Given that the RSA Cipher is an effective form of encoding integers, a method that can translate any text message into an integer and vice versa should be addressed. While there are multiple methods of translation, a particularly simple and efficient translation will be described below. First, each character will be assigned a numerical value. For example, $a = 0, b = 1, c = 2, d = 3, \cdots z = 25, [space] = 26$, etc. Note that this assignment is somewhat arbitrary. The only necessary qualities of this assignment are that each character is assigned a distinct integer and that the alphabet is known to both the sending and receiving parties. For the sake of simplicity, the assignment used in the following explanation will be as follows.

a	b	c	d	...	y	z	[space]	.	?	,	!
0	1	2	3	...	24	25	26	27	28	29	30

After an assignment of characters to integers is known by both parties, the process of converting a message such as "*hello there.*" into an integer falls to the task of compacting the list of integers representing each individual character into one long integer. In this case, "*hello there.*" translates to the list

$[7, 4, 11, 11, 14, 26, 19, 7, 4, 17, 4, 27]$. To translate this list into a single integer, we can employ a clever trick. We can multiply each number in the list by a successive power of the length of our alphabet, which in this case is 30. In this example, our new list would be $[7, 4(30), 11(30)^2, 11(30)^3, 14(30)^4, 26(30)^5, 19(30)^6, 7(30)^7, 4(30)^8, 17(30)^9,$ $4(30)^{10}, 27(30)^{11}]$. While we seem to have only made this list more complicated, we can simply add all of these terms together to result in one large integer that uniquely represents the entire list. In this case, $7 + 4(30) + 11(30)^2 + 11(30)^3 + 14(30)^4 + 26(30)^5 + 19(30)^6 + 7(30)^7 + 4(30)^8 + 17(30)^9 + 4(30)^{10} + 27(30)^{11} = 480996262984447027$. The key word to notice in the last two sentences was the word 'uniquely.' There is no other list in a 30-character alphabet that would collapse to that same integer. For proof, we can look at our own system of counting numbers. We start counting $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$, and then we create a '10s place' to get the number 10, which is our compacted way of writing $0 + 1(10)$, just as 11 is a compacted form of $1 + 1(10)$ and 378921 is a compacted form of $1 + 2(10) + 9(100) + 8(1000) + 7(10000) + 3(100000)$. Just as every integer in our everyday 10-character numbering system is unique, every integer resulting from our 30-character alphabet is also uniquely determined. No set of integers other than [1,8,3] when multiplied by successive powers of 10 can result in 381, and no set other than that corresponding to "*hello there.*" can result in 480996262984447027. Assuming the intended receiver knows which alphabet the sender is using, this integer can easily be decoded by using the process below, given an integer X and an alphabet of length L.

(1). Set a dummy variable $k = 1$

(2). Take $r \equiv X \mod L^k$

(3). Divide r/L^{k-1}.

(4). Append this result to a list containing the integers of this sequence.

(5). Reset X to be $X - r$

(6). Add 1 to k and repeat steps 2-4 until $X = 0$.

In this example, the receiver could take $480996262984447027 \equiv 7 \pmod{30}$, divide $7/30^0 = 7$, and know that the first integer of the list is 7. Then, the receiver could subtract $480996262984447027 - 7$, and take this $\pmod{900}$ to get a result of 120, which when divided by 30^1 yields 4. Our current list reads $[7, 4]$. Subtracting $480996262984447020 - 120$ and then applying the above steps repeatedly until we arrive at 0 will yield the rest of the sequence, which can then be translated into the alphabet.

5. Conclusion

We have now reviewed the entire process of encrypting and decrypting the RSA Cipher. Due to its being nearly impossible to crack without a private key, the method is extremely popular in modern cryptography, which only serves to further illustrate the subtle beauty of the underlying mathematics.

References

[1] Neal Koblitz, *A course in number theory and cryptography*, Volume 114, Springer Science & Business Media, (1994).

[2] Douglas Robert Stinson and Maura Paterson, *Cryptography: theory and practice*, CRC press, (2018).