GPH - International Journal of Mathematics
Journal homepage: www.gphjournal.org e-ISSN 2795-3278 p-ISSN 2795-3274
DOI: https://doi.org/10.5281/zenodo.10209900

NUMBER THEORY IN RSA ENCRYPTION SYSTEMS

Michael N. John[a] Ogoegbulem Ozioma[b] Obukohwo, Victor.[c] Henry Etaroghene Egbogho[d]

[a]Department of Mathematics, Akwa Ibom State University, Nigeria Email: storm4help1@gmail.com
[b]Department of Mathematics, Dennis Osadebay University, Anwai, asaba, Delta, State, Nigeria Email: Ozioma.ogoegbulem@dou.edu.ng
[c]Department of Mathematics, Dennis Osadebay University, Anwai, asaba, Delta State, Nigeria E: victor.obukohwo@dou.edu.ng
[d]Department of Mathematics, Dennis Osadebay University, Anwai, asaba, Delta State, Nigeria E: egbogho.henry@dou.edu.ng

A B S T R A C T

RSA encryption is a widely employed cryptographic system based on the principles of number theory. Number theory is the branch of pure mathematics that studies positive whole numbers, called natural numbers and integers. Number theory seeks to discover relationships existing between numbers. Cryptography is the study of encryption and decryption of information. By encrypting information, data can be transmitted to the intended recipient securely across the internet. In this paper, we propose an RSA encryption scheme based on number theory, showing its applications to cryptography.

Keywords:
RSA, Number Theory, Prime Numbers, Public Key Cryptography, Modular Arithmetic, Encryption, Decryption, Private Key, Security

1. INTRODUCTION

The RSA encryption algorithm relies on the intricate properties of number theory to provide secure communication over insecure channels. The fundamental premise involves the generation of large prime numbers, from which public and private keys are derived. In this paper, we will explain a widely used cryptographic algorithm known as RSA [7]. We will also look at the mathematics of Number Theory [8] which is necessary in the understanding of the working of RSA algorithm. We will implement our proposed algorithms with the computer language, Python and Java.

This paper is dedicated to exploring the principles of number theory and applying them to cryptography. Cryptography is an important aspect of secure communication which allows privacy between persons. It prevents interceptors or unlikely holders of your message from being able to read whatever you are sending to the endpoint. It protects information and a variety of information like passwords, keys, names, bank accounts, etc. This raises the question of what makes cryptography function, how cryptography works, and how number theory is linked to it.

* *Corresponding author.* Michael N. John.
E-mail address: storm4help1@gmail.com

The use of cryptography was originally meant for communicating with other people in wars or conflicts. This was used to prevent the other belligerents from reading messages which could give away large amounts of information. Today, cryptography is used to encrypt messages to prevent interceptors from reading them and is practiced all over the internet. We will be introducing mathematical techniques in number theory which can be further applied to cryptography in encrypting messages. We refer the reader to [1–3] for more details about number theory and [4–6] with [9-12] for more details about cryptography

2. DIVISIBILITY

Definition 2.1. A number a is said to be divisible by another number b (where $b \neq 0$) if there exists an integer q such that $a = b \times q$.

Definition 2.2. A common divisor of two integers a and b is an integer that divides both a and b.

Definition 2.3. The greatest common divisor of two integers a and b, denoted as gcd(a,b), is the greatest positive integer that divides both a and b.

Examples 2.4. For a=12 and b=18:

- Divisibility: 18 divides 12 since 12=18 × 0.6667.
- Common Divisors: 1,2,3,6 are common divisors of 12 and 18.
- GCD: gcd(12,18)=6gcd(12,18)=6.

Examples 2.5. For a=35 and b=70:

- Divisibility: 35 divides 70 since 70 = 35 × 2.
- Common Divisors: 1,5,7,35 are common divisors of 35 and 70.
- GCD: gcd(35,70) = 35.

Proposition 2.6. If a is divisible by b and b is divisible by c, then a is divisible by c.

Proof 2.6.1: Let's assume that a is divisible by b and b is divisible by c. We want to show that a is divisible by c.

By definition of divisibility, there exist integers q_1 and q_2 such that: $a = bq_1$ and $b = cq_2$

Now, we want to express a in terms of c. Substituting the second equation into the first, we get: $a = (cq_2)q_1$

Let $q = q_1q_2$, which is also an integer since it is the product of two integers. Now we have: $a = cq$

This shows that a is divisible by c because it can be expressed as cq for some integer q.

Theorem (Euclidean Algorithm) 2.7: For any two positive integers a and b, there exist unique non-negative integers q and r such that $a = bq + r$ and $0 \leq r < b$.

Proof 2.7.1. We will prove this theorem using the Well-Ordering Principle, which states that every non-empty set of non-negative integers has a smallest element.

Existence: Consider the set $S = \{a - bk \mid k \text{ is an integer and } a - bk \geq 0\}$. This set is non-empty because it contains at least one non-negative element, namely a. By the Well-Ordering Principle, S has a smallest element, say r. Therefore, there exists an integer q such that $a - bq = r$.

Now, let's examine r. If $r \geq b$, we can subtract b repeatedly until $r < b$, and we still stay within the set S because $r - b$ is also in S. This contradicts the minimality of r, so we must have $0 \leq r < b$.

Uniqueness: Assume there are two pairs of integers (q_1, r_1) and (q_2,r_2) such that $a = bq_1 + r_1$, $0 \leq r_1 < b$, $a = bq_2 + r_2$, and $0 \leq r_2 < b$.

Subtracting the two equations, we get $b(q_1 - q_2) = r_1 - r_2$. Since $0 \leq r_1, r_2 < b$, it follows that $-b < r_1 - r_2 < b$. But the only multiple of b in this range is 0. Therefore, $r_1 - r_2$ must be 0, implying that $q_1 - q_2 = 0$ as well. Hence, $q_1 = q_2$ and $r_1 = r_2$.

This proves the uniqueness of the pair (q, r), completing the proof of the theorem.

Theorem (Bézout's Identity) 2.8. For any integers a and b, there exist integers x and y such that $\gcd(a, b) = ax + by$.

Proof 2.8.1. Let S be the set of all positive integers of the form $ax+by$, where x and y are integers. By the Well-Ordering Principle, S has a smallest positive element, say d. We want to show that d is the greatest common divisor of a and b.

Existence: By the Euclidean Algorithm, we can express a as $a = bq + r$, where q is an integer and $0 \leq r < |b|$. Now, we can express r as $r = a - bq$.

This implies that $r = a - bq$ is a linear combination of a and b. But d is the smallest positive integer of the form $ax + by$, so r cannot be positive. The only way for r to be non-positive is if $r = 0$. Therefore, $a = bq$ and b divides a.

Since b divides a, and b divides itself, b is a common divisor of a and b.

Now, let c be any common divisor of a and b. By definition of divisibility, c divides a and c divides b, so c divides any linear combination of a and b. This means c divides r.

Since c divides both b and r, c divides any linear combination of b and r. In particular, c divides a.

Therefore, c is a common divisor of a and b and b is the greatest common divisor.

Uniqueness: Assume there are two pairs of integers (x_1, y_1) and (x_2, y_2) such that $ax_1 + by_1 = d$ and $ax_2 + by_2 = d$, where d is the greatest common divisor of a and b.

Subtracting the two equations, we get $a(x_1-x_2) + b(y_1-y_2) = 0$. Since d is the greatest common divisor, d divides both terms on the left side, so d divides $a(x_1-x_2)$ and d divides $b(y_1-y_2)$.

This implies that d divides a and d divides b, making d a common divisor of a and b. Since d is the greatest common divisor, d must be a divisor of any common divisor of a and b.

Therefore, d divides both $a(x_1-x_2)$ and $b(y_1-y_2)$, but it is also the greatest common divisor. This means d must divide any linear combination of a and b. In particular, d divides $a(x_1-x_2) + b(y_1-y_2)$.

Since $a(x_1-x_2) + b(y_1-y_2) = 0$, this implies that d divides 0, and the only positive integer that divides 0 is 1. Therefore, $d = 1$.

Hence, there exist unique integers x and y such that $ax + by = 1$, which is the definition of the greatest common divisor being 11. This completes the proof of Bézout's Identity.

Theorem (Fundamental Theorem of Arithmetic) 2.9. Every positive integer greater than 1 can be uniquely expressed as a product of prime numbers, up to the order of the factors.

Proof 2.9.1. Every positive integer greater than 1 can be uniquely expressed as a product of prime numbers, up to the order of the factors.

Base Case: Consider the smallest positive integer greater than 1, which is 2. 2 is prime, so it can be expressed as the product of itself.

Inductive Step: Assume that the statement is true for all positive integers from 2 to $n-1$, where $n > 1$. We want to show that it is true for n.

- If n is prime, then n is already a product of primes.
- If n is composite, then $n = ab$ for some positive integers a and b where $1 < a,b < n$. By the induction hypothesis, both a and b can be expressed as products of primes. Therefore, $n = ab$ can also be expressed as a product of primes.

Uniqueness:
Assume, for the sake of contradiction, that there exists a positive integer greater than 1 that can be expressed as the product of primes in two different ways, say $n=p_1 \cdot p_2 \cdot \ldots \cdot p_k$ and $n = q_1 \cdot q_2 \cdot \ldots \cdot q_m$, where p_i and q_i are prime numbers.
Without loss of generality, assume p_1 is not equal to any of the q_i. Since p_1 divides n, it must divide the product $q_1 \cdot q_2 \cdot \ldots \cdot q_m$. However, p_1 is prime, and by the Fundamental Theorem of Arithmetic, it must be one of the q_i.

This is a contradiction because we assumed that p_1 is not equal to any of the q_i. Therefore, there cannot be two different prime factorizations of n.

3.1 CONGRUENCES
Definition 3.1 Let a and b be two integers, and let m be a positive integer. We say that a is congruent to b modulus m, denoted by $a \equiv b \pmod{m}$, if m divides $a - b$.

Proof 3.1.1 We need to show that m divides $a-b$.

By definition, $a \equiv b \pmod{m}$ means that there exists an integer k such that $a - b = km$.

Rearranging the equation, we get $a = b + km$.

Now, let's express a as $b + km$, where k is an integer. Since k is an integer, m divides $a-b$, and therefore, $a \equiv b \pmod{m}$.

Example 3.2. Let's consider an example to illustrate congruence modulo m

Suppose we have a=17, b=3, and m=7. We want to check whether a is congruent to b modulo m, i.e., $17 \equiv 3 \pmod 7$.

The congruence holds because $17 - 3 = 14$ is divisible by $m = 7$. Specifically, $17 - 3 = 2 \times 7$.

Therefore, we can say that $17 \equiv 3 \pmod 7$.

4. SOLVING EQUATIONS

Example 4.1. Let's take an example: $2x \equiv 6 \pmod 5$.

To find x, we want to solve the equation $2x - 6 = 5k$ for some integer k. Rearranging, we get $2x = 5k + 6$.

A solution is $x = 8$, because $2 \times 8 - 6 = 10$, and 10 is divisible by 5.

Therefore, $2x \equiv 6 \pmod 5$ has a solution with $x = 8$.

Computational Code (Python) 4.2. Here is a simple Python code snippet that can be used to find a solution for $ax \equiv c \pmod m$

```
def find_x(a, c, m):
    for x in range(m):
        if (a * x) % m == c % m:
            return x
    return None
# Example: 2x ≡ 6 (mod 5)
a = 2
c = 6
m = 5
solution = find_x(a, c, m)
print(f"A solution for {a}x ≡ {c} (mod {m}) is x = {solution}")
```

This code defines a function **find_x** that iterates through possible values of x and returns the first one that satisfies the congruence equation. In the example, it finds that $2x \equiv 6 \pmod 5$ has a solution with x=8.

Example 4.2. Let p and q be two different prime numbers, and let $m = p\,q$. Let a be an integer such that $\gcd(a, m) = 1$. Then, $a^{(p-1)(q-1)} \equiv 1 \pmod m$. This is a special case of Euler's formula.

Illustration with Tables 4.2.1

Consider p=5 and q=7, so $m = pq = 35$. The totient function is $\phi(35) = (5-1)(7-1) = 24$

a	$a^{(p-1)(q-1)}$	mod m
2	2^{24}	1
3	3^{24}	1
4	4^{24}	1
5	5^{24}	1
6	6^{24}	1

Example 4.2.2. Let p=3 and q=11, so m=pq=33. Choose a=7, and verify Euler's formula:
$7^{(3-1)(11-1)} = 7^{20} \equiv 1 \pmod{33}$
This example demonstrates that $a^{(p-1)(q-1)}$ is congruent to 11 modulo m, consistent with Euler's theorem.

3. PROPOSED SCHEME

Creating a complete RSA encryption scheme involves various mathematical steps, including key generation, encryption, and decryption. We proposed a simplified outline along with Python and Java code snippets for key generation, encryption, and decryption.

Key Generation 5.1. In RSA, two large prime numbers, p and q, are generated. The modulus n is computed as $n = p \times q$, and the public exponent e and private exponent d are determined such that $e \times d \equiv 1 \pmod{\phi(n)}$, where $\phi(n)$ is Euler's totient function of n.

Python Code 5.1.1.

```
import random

def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            return False
    return True

def generate_prime(bits):
    while True:
```

```
        num = random.getrandbits(bits)
        if is_prime(num):
            return num

def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

def generate_keypair(bits):
    p = generate_prime(bits)
    q = generate_prime(bits)
    n = p * q
    phi_n = (p - 1) * (q - 1)

    e = random.randrange(1, phi_n)
    while gcd(e, phi_n) != 1:
        e = random.randrange(1, phi_n)

    d = mod_inverse(e, phi_n)

    return ((e, n), (d, n))

# Example usage:
public_key, private_key = generate_keypair(512)
print("Public Key:", public_key)
print("Private Key:", private_key)
```

Java Code 5.1.2.

```
import java.math.BigInteger;
import java.security.SecureRandom;

public class RSAKeyGenerator {

    private static SecureRandom random = new SecureRandom();

    private static BigInteger generatePrime(int bits) {
```

```
        BigInteger prime;
        do {
            prime = new BigInteger(bits, random);
        } while (!prime.isProbablePrime(100));
        return prime;
    }

    private static BigInteger modInverse(BigInteger a, BigInteger m) {
        return a.modInverse(m);
    }

    public static BigInteger[] generateKeyPair(int bits) {
        BigInteger p = generatePrime(bits);
        BigInteger q = generatePrime(bits);
        BigInteger n = p.multiply(q);
        BigInteger phi_n = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

        BigInteger e = new BigInteger(bits, random);
        while (e.compareTo(phi_n) >= 0 || e.gcd(phi_n).compareTo(BigInteger.ONE) != 0) {
            e = new BigInteger(bits, random);
        }

        BigInteger d = modInverse(e, phi_n);

        return new BigInteger[]{e, n, d, n};
    }

    public static void main(String[] args) {
        BigInteger[] keys = generateKeyPair(512);
        System.out.println("Public Key: " + keys[0] + " " + keys[1]);
        System.out.println("Private Key: " + keys[2] + " " + keys[3]);
    }
}
```

Encryption and Decryption 5.2. Once the keys are generated, you can use them for encryption and decryption.

Python Code 5.2.1.

```
def encrypt(message, public_key):
    e, n = public_key
    return pow(message, e, n)

def decrypt(ciphertext, private_key):
    d, n = private_key
```

```
    return pow(ciphertext, d, n)

# Example usage:
message = 42
ciphertext = encrypt(message, public_key)
decrypted_message = decrypt(ciphertext, private_key)
print("Original Message:", message)
print("Encrypted Message:", ciphertext)
print("Decrypted Message:", decrypted_message)
```

Java Code 5.2.2.

```
import java.math.BigInteger;

public class RSAEncryption {

    public static BigInteger encrypt(BigInteger message, BigInteger publicKey) {
        return message.modPow(publicKey, publicKey);
    }

    public static BigInteger decrypt(BigInteger ciphertext, BigInteger privateKey) {
        return ciphertext.modPow(privateKey, privateKey);
    }

    public static void main(String[] args) {
        BigInteger message = new BigInteger("42");
        BigInteger[] publicKey = {new BigInteger("65537"), new
BigInteger("119175209320792300284042168942030078229")};
        BigInteger[] privateKey = {new BigInteger("68506764043437774684643752074845734561"), new
BigInteger("119175209320792300284042168942030078229")};

        BigInteger encryptedMessage = encrypt(message, publicKey[0]);
        BigInteger decryptedMessage = decrypt(encryptedMessage, privateKey[0]);

        System.out.println("Original Message: " + message);
        System.out.println("Encrypted Message: " + encryptedMessage);
        System.out.println("Decrypted Message: " + decryptedMessage);
    }
}
```

The above codes are simplified for educational purposes and may not include additional optimizations or security measures used in real-world RSA implementations. Always use well-established libraries for cryptographic operations in actual applications.

6. CONCLUSION

RSA encryption system stands as a testament to the symbiotic relationship between cryptography and number theory. The foundational principles of prime numbers, modular arithmetic, and modular inverses form the bedrock of RSA's security. In this art paper we present the theory behind the widely used RSA algorithm in Cryptography. Cryptography is one of the many examples of the power of Mathematics. Cryptography has drastically evolved from its early days of simple ciphers, and the usage of number theory concepts make encryptions such as RSA secure even with the usage of computers. Number theory allows for stronger encryptions, and as computers get stronger, encryptions gets stronger. As quantum computers become prevalent, new encryptions grow to keep information safe from unwanted eyes.

7. ACKNOWLEDGEMENTS

This paper was written with the mentoring of Dr. Otobong G. Udoaka, Senior Lecturer, Department of Mathematics, Akwa Ibom State University, Nigeria.

REFERENCES

[1] K. Ireland and M. Rosen,A classical introduction to modern number theory, Second Edition, Springer-Verlag, New York,(1990).
[2] H. Stark,An introduction to number theory, Cambridge, The MIT Press, (1978).
[3] J. Silverman,A friendly introduction to number theory, Fourth Edition, Pearson, New York, (2011).
[4] S. Padhye, R. Sahu and V. Saraswat,Introduction to cryptography, Boca Raton, CRC Press, (2018).
[5] S. Rubinstein-Salzedo,Cryptography, Springer Nature, Switzerland, (2010).
[6] M. Omar,Number Theory Toward RSA Cryptography: in 10 Undergraduate Lectures (Discrete Mathematics), Volume 1,CreateSpace Independent Publishing, Scotts Valley, (2017).27
[7] Joseph H. Silverman,A Friendly Introduction to Number Theory, Am. Math. Compet., 10(2006), 12.
[8] Douglas Robert Stinson and Maura Paterson,Cryptography: theory and practice, CRC Press, (2018).
[9] Udoaka O. G. & Frank E. A. (2022). Finite Semi-group Modulo and Its Application to Symmetric Cryptography, International Journal of Pure Mathematics DOI: 10.46300/91019.2022.9.13.
[10] Udoaka, O. G. (2022). Generators and inner automorphism. THE COLLOQUIUM -A Multi-disciplinary Thematc Policy Journal www.ccsonlinejournals.com. Volume 10, Number 1 , Pages 102 -111 CC-BY-NC-SA 4.0 International Print ISSN : 2971-6624 eISSN: 2971-6632.
[11] Michael N. John & Udoaka O. G (2023). Algorithm and Cube-Lattice-Based Cryptography. International journal of Research Publication and reviews, Vol 4, no 10, pp 3312-3315 October 2023.
[12] Michael N. John, Udoaka O. G., "Computational GroupTheory and Quantum-Era Cryptography",International Journal of Scientific Research in Science,Engineering and Technology (IJSRSET), Online ISSN :2394-4099, Print ISSN : 2395-1990, Volume 10 Issue 6,pp. 01-10, November-December 2023. Available at doi :https://doi.org/10.32628/IJSRSET2310556