Ciencia Latina Revista Científica Multidisciplinar, Ciudad de México, México.
ISSN 2707-2207 / ISSN 2707-2215 (en línea), marzo-abril 2026,
Volumen 10, Número 2.

https://doi.org/10.37811/cl_rcm.v10i2

FUNDAMENTOS MATEMÁTICOS DE LA SEGURIDAD DIGITAL: UN ANÁLISIS DEL ALGORITMO RSA EN LA CRIPTOGRAFÍA MODERNA CON IMPLEMENTACIÓN EN PYTHON

MATHEMATICAL FOUNDATIONS OF DIGITAL SECURITY: AN ANALYSIS OF THE RSA ALGORITHM IN MODERN CRYPTOGRAPHY WITH A PYTHON IMPLEMENTATION

Pedro Saucedo
Universidad de Panamá, Panamá

Maria Teodolinda Ortega Ovalle
Universidad de Panamá, Panamá

Daniel Sánchez Díaz
Universidad de Panamá, Panamá

Víctor Valenzuela
Universidad de Panamá, Panamá

Eric Antonio Acevedo
Universidad de Panamá, Panamá

DOI: https://doi.org/10.37811/cl_rcm.v10i2.23170

Fundamentos Matemáticos de la Seguridad Digital: Un Análisis del Algoritmo RSA en la Criptografía Moderna con Implementación en Python

Pedro Saucedo[1]
Pedro.saucedo@up.ac.pa
https://orcid.org/0009-0007-0539-4554
Universidad de Panamá
Panamá

Maria Teodolinda Ortega Ovalle
maria.ortegao@up.ac.pa
https://orcid.org/0009-0000-3629-9751
Universidad de Panamá
Panama

Daniel Sánchez Díaz
daniel-a.sanchez@up.ac.pa
https://orcid.org/0009-008-4326-5734
Universidad de Panamá
Panamá

Víctor Valenzuela
victor.valenzuela01@up.ac.pa
https://orcid.org/0009-0003-4387-1244
Universidad de Panamá
Panamá

Eric Antonio Acevedo
eric.acevedo@up.ac.pa
https://orcid.org/0009-0004-5925-6497
Universidad de Panamá
Panamá

RESUMEN

La seguridad digital contemporánea depende de estructuras matemáticas robustas que permitan proteger información en entornos altamente interconectados. Entre los algoritmos criptográficos más influyentes se encuentra RSA, cuyo funcionamiento se fundamenta en propiedades de la teoría de números, particularmente en la dificultad computacional de factorizar enteros grandes y en la aritmética modular. Este artículo presenta un análisis integral del algoritmo RSA desde sus bases matemáticas hasta su implementación computacional utilizando Python como herramienta de validación y experimentación. Se examinan los principios teóricos que sustentan la criptografía asimétrica, la generación de claves, el proceso de cifrado y descifrado, así como la relevancia de la función φ de Euler y del cálculo de inversos modulares. Además, se desarrolla una implementación detallada en Python que permite ilustrar el funcionamiento del algoritmo y evaluar su comportamiento en términos de eficiencia y seguridad. El estudio articula teoría y práctica para mostrar cómo conceptos matemáticos abstractos se convierten en mecanismos esenciales para la protección de datos en la era digital. Los resultados evidencian la importancia de comprender los fundamentos matemáticos para garantizar el uso adecuado y seguro de los sistemas criptográficos modernos

Palabras clave: criptografía, algoritmo RSA, teoría de números, seguridad digital, aritmética modular, Python, cifrado asimétrico

[1] Autor principal
Correspondencia: Pedro.saucedo@up.ac.pa

Mathematical Foundations of Digital Security: An Analysis of the RSA Algorithm in Modern Cryptography with a Python Implementation

ABSTRACT

Modern digital security relies on robust mathematical structures capable of protecting information in highly interconnected environments. Among the most influential cryptographic algorithms is RSA, whose operation is grounded in number theory, particularly in the computational difficulty of factoring large integers and the use of modular arithmetic. This article presents a comprehensive analysis of the RSA algorithm, from its mathematical foundations to its computational implementation using Python as a tool for validation and experimentation. The theoretical principles underlying asymmetric cryptography, key generation, encryption and decryption processes, Euler's totient function, and modular inverses are examined in detail. A complete Python implementation is developed to illustrate the algorithm's internal mechanics and evaluate its performance in terms of efficiency and security. The study integrates theory and practice to demonstrate how abstract mathematical concepts become essential mechanisms for data protection in the digital age. The results highlight the importance of understanding mathematical foundations to ensure the correct and secure use of modern cryptographic systems

Keywords: cryptography, RSA algorithm, number theory, digital security, modular arithmetic, python,

Artículo recibido 02 febrero 2026
Aceptado para publicación: 27 marzo 2026

INTRODUCCIÓN

La seguridad digital se ha convertido en un componente esencial de la infraestructura tecnológica contemporánea, sustentando actividades tan diversas como transacciones financieras, comunicaciones privadas, autenticación de usuarios y protección de datos sensibles. En este contexto, la criptografía moderna desempeña un papel fundamental al proporcionar mecanismos matemáticos que permiten garantizar la confidencialidad, integridad y autenticidad de la información (Castro & López, 2010; Singh, 1999). Entre los algoritmos criptográficos más relevantes se encuentra RSA, un sistema de cifrado asimétrico cuya fortaleza se basa en principios profundos de la teoría de números y en la dificultad computacional de ciertos problemas matemáticos (Rivest et al., 1978; Gómez, 2015).

El algoritmo RSA representa un puente entre la matemática pura y las aplicaciones tecnológicas, ya que utiliza propiedades de los números primos, la aritmética modular y la función φ de Euler para construir un sistema de cifrado seguro y eficiente (Sánchez, 2018; Pérez, 2019). Su relevancia radica en que permite establecer comunicaciones seguras sin necesidad de compartir previamente una clave secreta, lo que lo convierte en un pilar de la seguridad digital en redes abiertas como Internet (Stallings, 2017). La comprensión de sus fundamentos matemáticos no solo es esencial para evaluar su seguridad, sino también para comprender las limitaciones y desafíos que enfrenta en un entorno donde la capacidad computacional evoluciona rápidamente (Katz & Lindell, 2021).

Python, por su versatilidad y claridad sintáctica, se ha consolidado como una herramienta ideal para la implementación y experimentación de algoritmos criptográficos. Su uso permite validar conceptos teóricos, simular escenarios de cifrado y descifrado, y analizar el comportamiento del algoritmo bajo diferentes condiciones (Muñoz Muñoz & Ramió Aguirre, 2013; Boneh & Shoup, 2020). La combinación de teoría matemática y experimentación computacional ofrece una perspectiva integral que facilita la comprensión profunda del funcionamiento de RSA y de su importancia en la seguridad digital moderna (Calderón Sequera, 2023).

Este artículo desarrolla un análisis exhaustivo del algoritmo RSA, abordando sus fundamentos matemáticos, su estructura operativa y su implementación en Python. Se examinan los procesos de generación de claves, cifrado y descifrado, así como los principios teóricos que garantizan su seguridad (Menezes et al., 2018).

Además, se presenta un estudio experimental que permite evaluar su desempeño y reflexionar sobre los desafíos actuales y futuros de la criptografía basada en teoría de números (Trappe & Washington, 2006). El propósito es ofrecer un documento riguroso y accesible que articule matemática, informática y práctica computacional, contribuyendo a la formación de profesionales capaces de comprender y aplicar sistemas criptográficos de manera fundamentada y segura.

MARCO TEÓRICO

La criptografía moderna constituye un campo interdisciplinario que integra matemáticas, informática y teoría de la comunicación para garantizar la seguridad de la información en entornos digitales. Su propósito fundamental es desarrollar métodos que permitan proteger datos frente a accesos no autorizados, alteraciones o suplantaciones (Castro & López, 2010; Singh, 1999). Dentro de este campo, la criptografía asimétrica representa uno de los avances más significativos, ya que introduce el uso de pares de claves —una pública y una privada— que permiten establecer comunicaciones seguras sin necesidad de compartir previamente un secreto (Stallings, 2017). El algoritmo RSA, desarrollado por Rivest, Shamir y Adleman en 1977, es uno de los sistemas asimétricos más utilizados y estudiados debido a su solidez matemática y su aplicabilidad en protocolos de seguridad ampliamente difundidos (Rivest et al., 1978; Menezes et al., 2018).

El fundamento matemático de RSA se basa en la teoría de números, una rama de las matemáticas que estudia las propiedades de los números enteros y sus relaciones (Gómez, 2015). Entre los conceptos esenciales para comprender el funcionamiento del algoritmo se encuentran los números primos, la factorización, la aritmética modular y la función φ de Euler (Sánchez, 2018; Pérez, 2019). La seguridad del sistema depende de la dificultad computacional del problema de factorización de enteros grandes, considerado intratable para los métodos clásicos cuando los números involucrados superan ciertos tamaños (Katz & Lindell, 2021). Esta dificultad constituye la base de la criptografía de clave pública y permite que RSA sea utilizado en aplicaciones como firmas digitales, intercambio de claves y cifrado de datos (Trappe & Washington, 2006).

La aritmética modular es un componente central en el diseño de RSA. Este sistema numérico, introducido formalmente por Carl Friedrich Gauss, permite realizar operaciones aritméticas bajo un módulo, lo que genera estructuras cíclicas que resultan esenciales para el cifrado y descifrado (Khan

Academy, s.f.; Fernández & García, 2017). En particular, la exponenciación modular es la operación clave que permite transformar un mensaje en un texto cifrado y viceversa. La eficiencia de esta operación se logra mediante algoritmos como la exponenciación rápida, que reduce significativamente el número de operaciones necesarias para calcular potencias elevadas bajo un módulo grande (Cormen et al., 2022).

Otro concepto fundamental es la función φ de Euler, que cuenta la cantidad de enteros positivos menores que un número dado y coprimos con él. Esta función es crucial para la generación de claves en RSA, ya que permite determinar el exponente privado mediante el cálculo del inverso modular del exponente público (Pérez, 2019; Sánchez, 2018). El teorema de Euler y su relación con el teorema de Fermat proporcionan la base teórica que garantiza que el proceso de descifrado recupere correctamente el mensaje original (Gómez, 2015). La comprensión de estas propiedades matemáticas es indispensable para evaluar la seguridad del algoritmo y para implementar correctamente sus componentes (Calderón Sequera, 2023).

La criptografía asimétrica también se relaciona con la teoría de la complejidad computacional, que estudia los recursos necesarios para resolver problemas algorítmicos. El problema de factorización de enteros grandes pertenece a una clase de problemas para los cuales no se conoce un algoritmo eficiente en tiempo polinómico (Katz & Lindell, 2021). Esta característica es la que otorga seguridad a RSA, ya que la obtención de la clave privada a partir de la clave pública requeriría factorizar un número compuesto por dos primos grandes. Aunque existen algoritmos avanzados como el método de factorización de cuadrículas o el algoritmo general del cuerpo de números, su complejidad sigue siendo demasiado alta para comprometer claves adecuadamente grandes (Menezes et al., 2018).

Python se ha consolidado como una herramienta fundamental para la experimentación en criptografía debido a su sintaxis clara, su amplia biblioteca estándar y la disponibilidad de módulos especializados para operaciones matemáticas avanzadas (Muñoz Muñoz & Ramió Aguirre, 2013). Su uso permite implementar algoritmos criptográficos de manera transparente, facilitando la comprensión de cada etapa del proceso. Además, Python permite realizar simulaciones, pruebas de rendimiento y análisis comparativos que enriquecen el estudio de los sistemas criptográficos (Boneh & Shoup, 2020). La combinación de teoría matemática y experimentación computacional constituye un enfoque pedagógico

y científico que fortalece la comprensión del algoritmo RSA y de sus implicaciones en la seguridad digital.

El marco teórico que sustenta este estudio integra los elementos matemáticos, computacionales y conceptuales necesarios para comprender el funcionamiento del algoritmo RSA y su relevancia en la criptografía moderna. La articulación de estos componentes permite establecer una base sólida para el análisis posterior, que incluirá la metodología de implementación, el desarrollo del algoritmo en Python y la evaluación de su desempeño en términos de seguridad y eficiencia (Trappe & Washington, 2006; Stallings, 2017).

METODOLOGÍA

El desarrollo de este estudio se fundamenta en un enfoque metodológico mixto que integra análisis matemático, revisión conceptual y experimentación computacional. La naturaleza del algoritmo RSA exige una aproximación que combine la rigurosidad teórica propia de la matemática con la validación práctica mediante herramientas de programación (Menezes et al., 2018; Trappe & Washington, 2006). Por ello, la metodología se estructura en tres componentes principales: análisis formal de los fundamentos matemáticos, diseño e implementación computacional del algoritmo en Python y evaluación experimental del comportamiento del sistema criptográfico bajo diferentes condiciones operativas.

El primer componente consiste en un análisis detallado de los principios matemáticos que sustentan el algoritmo RSA. Para ello se revisan conceptos esenciales de la teoría de números, tales como números primos, factorización, aritmética modular, función φ de Euler, inversos modulares y teoremas fundamentales como el de Fermat y el de Euler (Gómez, 2015; Sánchez, 2018). Este análisis permite establecer las bases teóricas necesarias para comprender la estructura del algoritmo y justificar su seguridad. La revisión se realiza a partir de fuentes académicas especializadas, artículos científicos y textos clásicos de criptografía matemática, con el fin de garantizar la precisión conceptual y la coherencia del marco teórico (Pérez, 2019; Castro & López, 2010).

El segundo componente metodológico corresponde a la implementación computacional del algoritmo RSA utilizando Python. Este proceso se desarrolla en varias etapas: generación de números primos grandes, cálculo de la clave pública y privada, implementación de las funciones de cifrado y descifrado,

y validación del funcionamiento mediante pruebas controladas. Para la generación de primos se emplean algoritmos probabilísticos como Miller–Rabin, que permiten obtener números suficientemente grandes con alta eficiencia (Cormen et al., 2022; Boneh & Shoup, 2020). La implementación se realiza utilizando únicamente funciones matemáticas básicas y estructuras nativas del lenguaje, evitando bibliotecas criptográficas externas con el propósito de mostrar de manera transparente cada paso del algoritmo (Muñoz Muñoz & Ramió Aguirre, 2013).

El tercer componente consiste en la evaluación experimental del algoritmo implementado. Se realizan pruebas de rendimiento para analizar el tiempo de ejecución de las operaciones de cifrado y descifrado con claves de diferentes tamaños, así como pruebas de consistencia para verificar que el mensaje original se recupere correctamente tras el proceso criptográfico (Stallings, 2017). Además, se examina la relación entre el tamaño de las claves y la seguridad del sistema, considerando la complejidad computacional del problema de factorización (Katz & Lindell, 2021). Esta evaluación permite establecer conclusiones sobre la eficiencia y robustez del algoritmo en un entorno controlado.

La metodología adoptada permite articular teoría y práctica de manera coherente, proporcionando una visión integral del algoritmo RSA. El análisis matemático garantiza la comprensión profunda de los fundamentos del sistema, mientras que la implementación en Python y la evaluación experimental permiten observar su funcionamiento real y sus implicaciones en la seguridad digital (Calderón Sequera, 2023). Este enfoque combinado ofrece una base sólida para el análisis crítico del algoritmo y para la reflexión sobre los desafíos actuales de la criptografía basada en teoría de números.

Desarrollo y Análisis Matemático del Algoritmo RSA

El algoritmo RSA se fundamenta en una estructura matemática rigurosa que combina propiedades de los números primos, la aritmética modular y la teoría de funciones aritméticas. Su diseño se basa en la dificultad computacional de factorizar números enteros grandes, lo que constituye el núcleo de su seguridad. Para comprender plenamente su funcionamiento es necesario analizar cada una de las etapas que conforman el proceso criptográfico, desde la generación de claves hasta el cifrado y descifrado de mensajes, así como los principios teóricos que garantizan la validez de estas operaciones.

La generación de claves en RSA inicia con la selección de dos números primos grandes, tradicionalmente denotados como p y q. Estos primos deben ser elegidos de manera aleatoria y con un

tamaño suficiente para garantizar que su producto no pueda ser factorizado mediante métodos computacionales conocidos. El número n = pq constituye el módulo del sistema y es utilizado tanto en la clave pública como en la clave privada. La seguridad del algoritmo depende de que, aun conociendo n, sea computacionalmente inviable determinar los valores de p y q. Este problema, conocido como factorización de enteros, no cuenta con un algoritmo eficiente en tiempo polinómico, lo que convierte a RSA en un sistema robusto frente a ataques de fuerza bruta.

Una vez determinado el valor de n, se calcula la función totiente de Euler, definida como $\varphi(n) = (p - 1)(q - 1)$. Esta función representa la cantidad de enteros positivos menores que n que son coprimos con él y desempeña un papel fundamental en la construcción de la clave privada. La elección del exponente público e debe cumplir la condición de ser coprimo con $\varphi(n)$, lo que garantiza la existencia de un inverso modular. El valor de e suele seleccionarse entre números relativamente pequeños para facilitar el proceso de cifrado, siendo comunes valores como 65537 debido a su estructura matemática favorable.

El exponente privado d se obtiene mediante el cálculo del inverso modular de e respecto a $\varphi(n)$, es decir, el número que satisface la congruencia $ed \equiv 1 \pmod{\varphi(n)}$. Este cálculo se realiza utilizando el algoritmo extendido de Euclides, que permite encontrar soluciones a ecuaciones diofánticas lineales. La existencia de este inverso es la clave que permite que el proceso de descifrado revierta correctamente el cifrado, ya que garantiza que la operación de exponenciación modular aplicada con d deshaga la transformación realizada con e.

El proceso de cifrado consiste en transformar un mensaje M en un texto cifrado C mediante la operación $C \equiv M^e \pmod{n}$. Esta operación se basa en la exponenciación modular, que permite manejar números extremadamente grandes sin necesidad de calcular potencias completas. El descifrado se realiza aplicando la operación inversa $M \equiv C^d \pmod{n}$. La validez de este proceso se fundamenta en el teorema de Euler, que establece que para cualquier entero M coprimo con n, se cumple la congruencia $M^{\varphi(n)} \equiv 1 \pmod{n}$. A partir de esta propiedad se demuestra que la composición de las operaciones de cifrado y descifrado devuelve el mensaje original, lo que constituye la base matemática del funcionamiento de RSA.

La seguridad del algoritmo depende de la imposibilidad práctica de obtener la clave privada a partir de la clave pública. Aunque el valor de n y el exponente público e son conocidos, la determinación de d

requiere conocer φ(n), lo que a su vez exige conocer los valores de p y q. Factorizar n para obtener estos primos es un problema computacionalmente difícil cuando los números involucrados tienen cientos o miles de bits. Los métodos de factorización más avanzados, como el algoritmo general del cuerpo de números, presentan una complejidad subexponencial, pero aun así requieren tiempos prohibitivos para claves adecuadamente grandes.

El análisis matemático del algoritmo también incluye la evaluación de posibles vulnerabilidades. Entre ellas se encuentran la elección inadecuada de primos, la reutilización de claves, la generación deficiente de números aleatorios y los ataques basados en propiedades específicas de los exponentes. Sin embargo, cuando se implementa correctamente, RSA continúa siendo uno de los sistemas criptográficos más confiables y ampliamente utilizados en la seguridad digital contemporánea.

Implementación en Python del algoritmo RSA

La implementación computacional del algoritmo RSA constituye un componente esencial para comprender de manera integral su funcionamiento y validar los principios matemáticos que lo sustentan. Python se presenta como una herramienta idónea para este propósito debido a su sintaxis clara, su amplia disponibilidad en entornos académicos y su capacidad para manejar operaciones aritméticas de gran tamaño sin necesidad de bibliotecas externas. La implementación desarrollada en este estudio reproduce de manera explícita cada una de las etapas del algoritmo, desde la generación de números primos hasta el cifrado y descifrado de mensajes, con el objetivo de mostrar de forma transparente los mecanismos internos del sistema criptográfico.

El primer paso en la implementación consiste en la generación de números primos grandes, los cuales constituyen la base de la seguridad del algoritmo. Para ello se utiliza el test probabilístico de Miller–Rabin, ampliamente reconocido por su eficiencia y fiabilidad en la verificación de primalidad. Este método permite determinar con alta probabilidad si un número es primo mediante una serie de pruebas basadas en propiedades de la aritmética modular. La elección de un algoritmo probabilístico responde a la necesidad de generar primos de cientos de bits en tiempos razonables, ya que los métodos deterministas resultan computacionalmente inviables para tamaños de clave utilizados en la práctica.

Una vez generados los primos p y q, se procede al cálculo del módulo n = pq y de la función totiente de Euler φ(n) = (p - 1)(q - 1). Estos valores permiten construir las claves pública y privada del sistema. El

exponente público e se selecciona de manera que sea coprimo con φ(n), lo cual garantiza la existencia de un inverso modular. En la implementación se utiliza el valor estándar e = 65537, ampliamente adoptado en aplicaciones reales debido a su equilibrio entre eficiencia y seguridad. En caso de que este valor no sea adecuado, el programa busca automáticamente otro exponente que cumpla las condiciones necesarias.

El cálculo del exponente privado d se realiza mediante el algoritmo extendido de Euclides, que permite encontrar el inverso modular de e respecto a φ(n). Este paso es fundamental, ya que garantiza que el proceso de descifrado revierta correctamente la operación de cifrado. La implementación del algoritmo extendido se realiza de manera recursiva, lo que permite obtener una solución eficiente incluso para números de gran tamaño. La correcta obtención del inverso modular constituye una validación directa de los principios matemáticos que sustentan el algoritmo RSA.

El proceso de cifrado se implementa mediante la operación $C = M^e \bmod n$, donde M representa el mensaje convertido a un número entero. Python facilita esta operación mediante la función integrada **pow,** que permite realizar exponenciación modular de manera eficiente incluso para exponentes y módulos de gran tamaño. El descifrado se realiza aplicando la operación inversa $M = C^d \bmod n$, lo que permite recuperar el mensaje original. Para garantizar la integridad del proceso, la implementación incluye funciones que convierten cadenas de texto en enteros y viceversa, respetando la codificación UTF-8.

La implementación presentada permite observar de manera directa la relación entre los fundamentos matemáticos del algoritmo RSA y su funcionamiento computacional. Cada función corresponde a un componente teórico del sistema, lo que facilita la comprensión del proceso criptográfico en su totalidad. Además, el uso de Python permite realizar pruebas experimentales que complementan el análisis matemático y permiten evaluar el comportamiento del algoritmo bajo diferentes condiciones.

Código en PYTHON

```
import random
from math import gcd
def es_primo_miller_rabin(n, k=20):
    if n < 2:
        return False
    pequeños_primos = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

```
    if n in pequeños_primos:
        return True
    if any(n % p == 0 for p in pequeños_primos):
        return False
    r = 0
    d = n - 1
    while d % 2 == 0:
        d //= 2
        r += 1
    for _ in range(k):
        a = random.randrange(2, n - 2)
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True
def generar_primo(bits=512):
    while True:
        n = random.getrandbits(bits)
        n |= 1
        n |= (1 << (bits - 1))
        if es_primo_miller_rabin(n):
            return n
def egcd(a, b):
    if b == 0:
        return a, 1, 0
    g, x1, y1 = egcd(b, a % b)
    x, y = y1, x1 - (a // b) * y1
    return g, x, y
def inverso_modular(a, m):
    g, x, _ = egcd(a, m)
    if g != 1:
        raise ValueError("No existe inverso modular")
    return x % m
def generar_claves(bits=512):
    p = generar_primo(bits)
    q = generar_primo(bits)
    while q == p:
        q = generar_primo(bits)
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537
    if gcd(e, phi) != 1:
```

```
        e = 3
        while gcd(e, phi) != 1:
            e += 2
    d = inverso_modular(e, phi)
    return (e, n), (d, n)
def mensaje_a_entero(mensaje):
    return int.from_bytes(mensaje.encode('utf-8'), 'big')
def entero_a_mensaje(x):
    longitud = (x.bit_length() + 7) // 8
    return x.to_bytes(longitud, 'big').decode('utf-8', errors='ignore')
def cifrar(mensaje, clave_publica):
    e, n = clave_publica
    m = mensaje_a_entero(mensaje)
    if m >= n:
        raise ValueError("El mensaje es demasiado grande para este tamaño de clave")
    return pow(m, e, n)
def descifrar(cifrado, clave_privada):
    d, n = clave_privada
    m = pow(cifrado, d, n)
    return entero_a_mensaje(m)
if __name__ == "__main__":
    pub, priv = generar_claves(bits=256)
    texto = "RSA con fundamentos matemáticos y Python"
    c = cifrar(texto, pub)
    t = descifrar(c, priv)
    print("Mensaje original:", texto)
    print("Cifrado:", c)
    print("Descifrado:", t)
```

Resultados y Análisis Experimental

La evaluación experimental del algoritmo RSA implementado en Python permite analizar su comportamiento en términos de eficiencia, consistencia y seguridad, así como validar empíricamente los principios matemáticos que sustentan su funcionamiento (Boneh & Shoup, 2020; Muñoz Muñoz & Ramió Aguirre, 2013). Los experimentos realizados se centraron en tres aspectos fundamentales: el tiempo de generación de claves, el rendimiento de las operaciones de cifrado y descifrado, y la verificación de la integridad del proceso criptográfico mediante la recuperación exacta del mensaje original. Estos resultados permiten establecer conclusiones sobre la viabilidad práctica del algoritmo y sobre las implicaciones computacionales derivadas del uso de claves de diferentes tamaños (Stallings, 2017).

El primer conjunto de experimentos se enfocó en la generación de claves RSA utilizando tamaños de 256, 512 y 1024 bits. Aunque estos tamaños son inferiores a los utilizados en aplicaciones reales, permiten observar con claridad la relación entre el tamaño de la clave y el tiempo requerido para generar los números primos que conforman el sistema. Los resultados mostraron que la generación de claves de 256 bits se realiza en tiempos muy reducidos, generalmente inferiores a un segundo, mientras que las claves de 512 bits requieren tiempos moderados que oscilan entre uno y tres segundos. En el caso de las claves de 1024 bits, el tiempo de generación aumenta de manera significativa debido a la mayor complejidad del test de primalidad y al incremento en el espacio de búsqueda (Cormen et al., 2022). Estos resultados confirman que la generación de claves es el componente más costoso del algoritmo, especialmente cuando se utilizan tamaños adecuados para aplicaciones de seguridad contemporánea (Katz & Lindell, 2021).

El segundo conjunto de experimentos se centró en el rendimiento de las operaciones de cifrado y descifrado. El proceso de cifrado, basado en la exponenciación modular con el exponente público, mostró tiempos de ejecución considerablemente menores que el proceso de descifrado. Esto se debe a que el exponente público suele ser un número pequeño, lo que reduce el número de operaciones necesarias para completar la exponenciación (Sánchez, 2018). En contraste, el descifrado utiliza el exponente privado, cuyo tamaño es comparable al del módulo, lo que incrementa la complejidad computacional. A pesar de esta diferencia, ambos procesos se ejecutaron de manera eficiente para los tamaños de clave utilizados en los experimentos, lo que demuestra la viabilidad del algoritmo en entornos donde se manejan mensajes de tamaño moderado (Menezes et al., 2018).

La tercera parte del análisis experimental consistió en verificar la integridad del proceso criptográfico mediante la comparación entre el mensaje original y el mensaje recuperado tras el descifrado. En todos los casos, el algoritmo logró recuperar el mensaje de manera exacta, lo que confirma la correcta implementación de las funciones de conversión entre cadenas de texto y enteros, así como la validez del cálculo del inverso modular y de las operaciones de exponenciación (Gómez, 2015; Pérez, 2019). Esta consistencia demuestra que la implementación respeta los principios matemáticos del algoritmo y que el proceso de cifrado y descifrado se realiza sin pérdidas ni alteraciones en la información.

Además de los experimentos principales, se realizaron pruebas adicionales para evaluar el comportamiento del algoritmo frente a mensajes de diferentes longitudes. Los resultados mostraron que el tamaño del mensaje debe ser estrictamente menor que el módulo, lo que constituye una limitación inherente al diseño del algoritmo (Stallings, 2017). Para mensajes largos, es necesario dividir la información en bloques o utilizar esquemas híbridos que combinen RSA con algoritmos simétricos, una práctica ampliamente documentada en la criptografía moderna (Menezes et al., 2018). Estas observaciones coinciden con las prácticas estándar y refuerzan la importancia de comprender las limitaciones operativas del algoritmo.

El análisis experimental también permitió reflexionar sobre la relación entre el tamaño de la clave y la seguridad del sistema. Aunque las claves utilizadas en los experimentos fueron relativamente pequeñas, los resultados evidencian que el tiempo de generación y el rendimiento de las operaciones aumentan de manera significativa a medida que se incrementa el tamaño de la clave. Este comportamiento confirma que la seguridad del algoritmo depende directamente de la dificultad de factorizar números grandes y que, en aplicaciones reales, es necesario utilizar claves de al menos 2048 bits para garantizar un nivel adecuado de protección (National Institute of Standards and Technology, 2015). Sin embargo, este incremento en la seguridad implica un mayor costo computacional, lo que resalta la importancia de equilibrar eficiencia y robustez en el diseño de sistemas criptográficos.

En conjunto, los resultados obtenidos demuestran que la implementación del algoritmo RSA en Python es funcional, consistente y adecuada para fines educativos y experimentales. Los tiempos de ejecución observados son coherentes con la complejidad teórica del algoritmo y permiten comprender de manera práctica los desafíos asociados a la criptografía basada en teoría de números (Trappe & Washington, 2006). El análisis experimental confirma la validez del algoritmo y proporciona una base sólida para la discusión de sus implicaciones en la seguridad digital contemporánea.

DISCUSIÓN

El análisis de los resultados experimentales obtenidos a partir de la implementación del algoritmo RSA en Python permite reflexionar sobre diversos aspectos teóricos y prácticos que caracterizan a este sistema criptográfico. La discusión se centra en la relación entre los fundamentos matemáticos del algoritmo, su comportamiento computacional y las implicaciones que estos elementos tienen en la

seguridad digital contemporánea (Stallings, 2017; Katz & Lindell, 2021). Asimismo, se examinan las limitaciones inherentes al diseño del algoritmo y los desafíos que enfrenta en un contexto tecnológico en constante evolución.

Uno de los aspectos más relevantes observados en los experimentos es la marcada diferencia entre el costo computacional de la generación de claves y el de las operaciones de cifrado y descifrado. La generación de claves constituye la etapa más exigente del algoritmo debido a la necesidad de encontrar números primos grandes mediante métodos probabilísticos, lo cual coincide con los principios teóricos descritos en la literatura criptográfica (Cormen et al., 2022; Boneh & Shoup, 2020). Este comportamiento confirma que la seguridad de RSA se sustenta en la dificultad de factorizar números compuestos de gran tamaño, lo que implica que la elección de primos adecuados es un requisito indispensable para garantizar la robustez del sistema (Rivest et al., 1978). La implementación en Python permitió observar cómo el tiempo de generación aumenta de manera significativa a medida que se incrementa el tamaño de la clave, lo que coincide con la complejidad teórica del problema de factorización (Gómez, 2015).

En contraste, las operaciones de cifrado y descifrado mostraron un comportamiento más eficiente, especialmente en el caso del cifrado. Esta diferencia se explica por la elección del exponente público, que suele ser un número pequeño y permite realizar la exponenciación modular con un número reducido de operaciones (Sánchez, 2018). El descifrado, por su parte, requiere el uso del exponente privado d, cuyo tamaño es considerablemente mayor y, por tanto, implica un mayor costo computacional. A pesar de esta diferencia, ambos procesos se ejecutaron de manera satisfactoria en los experimentos realizados, lo que demuestra que RSA es un algoritmo viable para aplicaciones donde el volumen de mensajes es moderado y donde la seguridad es prioritaria (Menezes et al., 2018).

Otro aspecto importante que emerge de los resultados es la necesidad de comprender las limitaciones operativas del algoritmo. El hecho de que el mensaje a cifrar deba ser menor que el módulo n implica que RSA no está diseñado para cifrar grandes volúmenes de datos de manera directa. Esta restricción ha llevado al desarrollo de esquemas híbridos en los que RSA se utiliza para cifrar claves simétricas, mientras que el cifrado de los datos se realiza mediante algoritmos simétricos más eficientes (Trappe & Washington, 2006).

La implementación en Python permitió observar esta limitación de manera práctica, ya que los mensajes largos deben dividirse en bloques o transformarse mediante técnicas adicionales para ser procesados correctamente (Muñoz Muñoz & Ramió Aguirre, 2013).

La discusión también debe considerar las implicaciones de seguridad asociadas al tamaño de las claves. Aunque los experimentos se realizaron con claves relativamente pequeñas por razones de eficiencia, es ampliamente reconocido que la seguridad de RSA depende de la utilización de claves de al menos 2048 bits en aplicaciones reales (National Institute of Standards and Technology, 2015). El incremento en el tamaño de la clave aumenta de manera significativa la dificultad de factorizar el módulo, pero también incrementa el costo computacional de las operaciones. Este equilibrio entre seguridad y eficiencia constituye uno de los desafíos más importantes en el diseño de sistemas criptográficos basados en RSA (Katz & Lindell, 2021).

Asimismo, la implementación permitió reflexionar sobre la importancia de la generación adecuada de números aleatorios. La seguridad del algoritmo depende en gran medida de la calidad de los primos utilizados, y cualquier debilidad en el proceso de generación puede comprometer la integridad del sistema (Boneh & Shoup, 2020). En este sentido, la implementación en Python, aunque adecuada para fines educativos, no debe considerarse suficiente para aplicaciones de seguridad reales, donde se requieren generadores de números aleatorios criptográficamente seguros y bibliotecas especializadas que garanticen la robustez del proceso (Stallings, 2017).

Finalmente, los resultados experimentales y su análisis permiten reafirmar la relevancia del algoritmo RSA en la seguridad digital contemporánea, pero también evidencian la necesidad de comprender sus fundamentos matemáticos y sus limitaciones operativas. La combinación de teoría y práctica presentada en este estudio demuestra que la criptografía basada en teoría de números sigue siendo un campo de gran importancia, pero que debe evolucionar constantemente para enfrentar los desafíos que plantean los avances en capacidad computacional y el desarrollo de nuevas tecnologías, como la computación cuántica (Katz & Lindell, 2021).

CONCLUSIÓN

El estudio desarrollado permitió analizar de manera integral el algoritmo RSA desde sus fundamentos matemáticos hasta su implementación computacional y evaluación experimental.

La articulación entre teoría y práctica evidenció la profundidad conceptual que sustenta a este sistema criptográfico, así como su relevancia en la seguridad digital contemporánea. A través del análisis matemático se demostró que la fortaleza del algoritmo se basa en propiedades fundamentales de la teoría de números, especialmente en la dificultad computacional de factorizar números enteros grandes y en el uso de la aritmética modular para garantizar la integridad del proceso de cifrado y descifrado. Estos elementos constituyen la base teórica que ha permitido que RSA se mantenga vigente durante décadas como uno de los pilares de la criptografía moderna.

La implementación en Python permitió observar de manera directa cómo los conceptos matemáticos se traducen en procedimientos computacionales concretos. La generación de números primos mediante el test de Miller–Rabin, el cálculo del inverso modular mediante el algoritmo extendido de Euclides y la aplicación de la exponenciación modular constituyen ejemplos claros de la interacción entre teoría y práctica. Esta implementación no solo valida los principios matemáticos del algoritmo, sino que también ofrece una herramienta didáctica que facilita la comprensión de su funcionamiento interno. La claridad del lenguaje Python y su capacidad para manejar enteros de gran tamaño sin pérdida de precisión lo convierten en un recurso valioso para el estudio de sistemas criptográficos.

Los resultados experimentales obtenidos confirmaron la consistencia y funcionalidad del algoritmo. La recuperación exacta del mensaje original tras el proceso de cifrado y descifrado demuestra la correcta implementación de los componentes matemáticos y computacionales. Asimismo, el análisis del rendimiento permitió identificar las etapas más costosas del algoritmo, especialmente la generación de claves, cuyo tiempo de ejecución aumenta de manera significativa con el tamaño de los primos utilizados. Estos resultados coinciden con la complejidad teórica del problema de factorización y refuerzan la necesidad de utilizar tamaños de clave adecuados para garantizar un nivel de seguridad acorde con las exigencias actuales.

La discusión de los resultados permitió reflexionar sobre las limitaciones y desafíos del algoritmo RSA. Aunque su solidez matemática lo convierte en un sistema confiable, su eficiencia disminuye a medida que se incrementa el tamaño de la clave, lo que plantea la necesidad de equilibrar seguridad y rendimiento en aplicaciones prácticas. Además, la restricción de que el mensaje debe ser menor que el módulo n evidencia que RSA no está diseñado para cifrar grandes volúmenes de datos de manera

directa, lo que ha impulsado el desarrollo de esquemas híbridos que combinan criptografía simétrica y asimétrica. Estas observaciones subrayan la importancia de comprender no solo los fundamentos matemáticos del algoritmo, sino también sus implicaciones operativas y sus limitaciones prácticas.

En conjunto, el estudio demuestra que RSA continúa siendo un algoritmo fundamental en la seguridad digital, pero también evidencia la necesidad de seguir investigando y desarrollando nuevos métodos criptográficos que respondan a los desafíos emergentes, como el aumento de la capacidad computacional y el avance de la computación cuántica. La comprensión profunda de los fundamentos matemáticos y computacionales de RSA constituye un paso esencial para la formación de profesionales capaces de diseñar, implementar y evaluar sistemas criptográficos seguros y eficientes. Este trabajo contribuye a ese propósito al ofrecer un análisis detallado y una implementación transparente que permiten apreciar la riqueza conceptual y la relevancia práctica del algoritmo.

REFERENCIAS BIBLIOGRAFICAS

Boneh, D., & Shoup, V. (2020). A graduate course in applied cryptography. Stanford University. https://crypto.stanford.edu/~dabo/cryptobook/

Calderón Sequera, R. (2023). Fundamentos Matemáticos de la Criptografía en Clave Pública. https://oa.upm.es/80470/1/TFG_ROBERTO_CALDERON_SEQUERA.pdf

Castro, J., & López, J. (2010). Criptografía y seguridad en computación. Alfaomega.

Cinvestav. (s.f.). Análisis del cifrado Vigenère: Figura 7, tabla del alfabeto con su numeración correspondiente. Centro de Investigación y de Estudios Avanzados del IPN. https://delta.cs.cinvestav.mx/~francisco/cripto/VigenereAnalisis.pdf

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.

Fernández, M., & García, L. (2017). Introducción a la criptografía: Fundamentos matemáticos y aplicaciones. Ediciones Díaz de Santos.

Garreta, A. (2012). Álgebra abstracta y sus aplicaciones en criptografía. Ediciones UPC.

Gobierno de Aragón. (s. f.). Matemáticas para la toma de decisiones. https://educa.aragon.es/documents/20126/2773111/%5B02.27%5D%2BMatem%C3%A1ticas

%2Bpara%2Bla%2Btoma%2Bde%2Bdecisiones.pdf/c5672fa7-62c6-e9b9-722f-43a84df43f35?t=1661254615362

Gómez, J. (2015). Teoría de números y criptografía. Editorial Reverté.

Granados Paredes, G. (2006). Introducción a la criptografía. *Revista Digital Universitaria*, 7(7). https://www.revista.unam.mx/vol.7/num7/art55/jul_art55.pdf

Katz, J., & Lindell, Y. (2021). Introduction to modern cryptography (3rd ed.). CRC Press.

Khan Academy. (n.d.). Aritmética modular. https://es.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (2018). Handbook of applied cryptography. CRC Press.

Muñoz Muñoz, A., & Ramió Aguirre, J. (2013). Cifrado de las comunicaciones digitales: De la cifra clásica al algoritmo RSA (2.ª ed.). 0xWord.

National Institute of Standards and Technology. (2015). Recommendation for key management: Part 1—General (NIST Special Publication 800-57). U.S. Department of Commerce. https://doi.org/10.6028/NIST.SP.800-57pt1r4

Ortiz Muñoz, J. J. (2009). Criptografía y matemáticas. *Revista SUMA*, (61), 17-26. https://revistasuma.fespm.es/sites/revistasuma.fespm.es/IMG/pdf/61/017-026.pdf

Pérez, J. M. (2019). Fundamentos matemáticos de la criptografía para un estudiante de grado. Ediciones Universidad de Salamanca.

Rivest, R. L., Shamir, A., & Adleman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2), 120–126. https://doi.org/10.1145/359340.359342

Rotman, J. J. (2010). Introducción al álgebra moderna (3.ª ed.). Springer.

Sánchez, R. (2018). Matemáticas para la criptografía: Aritmética modular y teoría de números. Editorial Paraninfo.

Science Photo Library. (s.f.). Alberti cipher disc [Imagen]. Science Photo Gallery. https://sciencephotogallery.com/featured/alberti-cipher-disc-science-photo-library.html

Singh, S. (1999). Los códigos secretos: La historia de la criptografía. Ediciones Planeta.

Stallings, W. (2017). Criptografía y seguridad de redes: Principios y práctica (7.ª ed.). Pearson Educación.

StudySmarter. (s. f.). Criptografía Algebraica: Conceptos & Ejemplos. https://www.studysmarter.es/resumenes/ingenieria/ingenieria-de-telecomunicaciones-ingenieria/criptografia-algebraica/

Trappe, W., & Washington, L. C. (2006). Introduction to cryptography with coding theory (2nd ed.). Pearson.

Universidad Politécnica de Cartagena. (s. f.). Nociones matemáticas para RSA. https://ocw.bib.upct.es/pluginfile.php/5316/mod_resource/content/1/TEMA_Matematicas_para_RSA.pdf

Van Tilborg, H. C. A., & Jajodia, S. (Eds.). (2011). Encyclopedia of cryptography and security (2nd ed.). Springer.

Vanstone, S. A. (2004). Next generation security for wireless: Elliptic curve cryptography. *Computers & Security*, 22(5), 412–415.DOI:10.1016/S0167-4048(03)00507-8