Advanced Techniques for Prime Number Generation in RSA Encryption: A Path Toward Greater Efficiency

Kwame Assa-Agyei
Department of Computer Science
Nottingham Trent University
Nottingham, United Kingdom
kwame.assa-agyei@ntu.ac.uk

Kayode Owa
Department of Computer Science
Nottingham Trent University
Nottingham, United Kingdom
kayode.owa@ntu.ac.uk

Tawfik Al-Hadhrami
Department of Computer Science
Nottingham Trent University
Nottingham, United Kingdom
tawfik.al-hadhrami@ntu.ac.uk

Seth Kwame Asafo
Department of Information Technology
National Information Technology Agency
Accra, Ghana
seth.asafo@nita.gov.gh

***Abstract*—** Cryptography is essential for securing communication and information in the digital age, particularly with the rise of the Internet of Things (IoT), which increases the risk of unauthorized data access and manipulation. Public key cryptography, specifically RSA encryption, plays a critical role in ensuring secure data transmission. However, RSA faces several challenges, including performance issues with key generation, encryption, and decryption times, especially as key sizes increase. It is also vulnerable to attacks like the Greatest Common Divisor (GCD) attack. This study aims to enhance the efficiency of cryptographic algorithms by evaluating standard RSA and proposing an optimized variant of RSA. The proposed variant incorporates prime sieving, cache reuse, and early termination in the Miller-Rabin primality test. Using a dataset of 50 randomly generated numbers between 1000 and 99999999999 both algorithms were tested 50 times to assess key metrics such as key generation time, encryption time, decryption time, and memory usage across key sizes of 1024, 2048, and 4096 bits. The results demonstrate that the proposed RSA variant significantly improves encryption and decryption times, along with reduced memory usage, while maintaining performance consistency as key sizes grow.

Keywords— Cryptography; Public key; RSA; memory; encryption; decryption

I. INTRODUCTON

Public key cryptography has played a transformative role in securing digital communication, enabling the confidential and authenticated exchange of information across insecure channels. As technology continues to advance, so do the associated risks of data breaches, online fraud, and cybercrime [1]. In this context, RSA encryption remains a cornerstone of secure data transmission, protecting sensitive information from malicious actors. The increasing reliance on digital records in government, corporate, and personal settings has further emphasized the importance of strong cryptographic measures. Electronic records offer advantages such as improved accessibility, efficiency, and reduced physical storage requirements, but they also demand heightened security measures to prevent unauthorized access and ensure compliance with privacy standards [2]. RSA encryption, as a key part of modern security protocols, is crucial in safeguarding these digital systems.

Despite its effectiveness, RSA encryption faces several performance challenges, particularly as key sizes grow larger. These challenges include slow key generation, encryption, and decryption times, which become more pronounced as the size of the prime numbers used increases. Moreover, RSA's reliance on prime numbers for key generation makes it susceptible to certain attacks, such as those exploiting the Greatest Common Divisor (GCD). Although various RSA variants, such as Batch RSA, Mprime RSA, Mpower RSA, and Rebalanced RSA, have been proposed to improve decryption speed, they do not fully address the performance bottlenecks inherent in larger prime sizes [3]. This study aims to optimize the computational efficiency of the RSA encryption scheme by enhancing key generation, encryption, and decryption processes while maintaining strong security. The key contributions of the study are as follows:

i. Improved Performance: The use of techniques such as prime sieving, cache reuse, and early termination in primality testing significantly reduces the time required for generating primes.

ii. Increased Efficiency: By leveraging precomputed primes and cache optimization, the computational overhead is minimized, leading to improved process scalability and overall efficiency.

The paper is organized as follows: Section II presents a literature review, and Section III outlines the experimental setup. Sections IV and V introduce the proposed RSA algorithm and discuss the performance

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

results respectively. Finally, the conclusion is presented in section VI.

II. RELATED WORK

The generation of prime numbers is a pivotal aspect of RSA encryption, which is widely used in public key cryptography. Maurer (1995) introduced a recursive algorithm designed to expedite the generation of prime numbers crucial for RSA. This algorithm enhanced the speed of producing random provable primes by exploiting the density and distribution characteristics of primes, though it highlighted the potential for further optimization in large cryptographic applications [4]. Building on this, the author in [5] addressed the computational limitations in prime number generation, particularly for smart card implementations. Their proposed algebraic methods significantly reduced the complexity of the algorithms, enhancing both memory and computational efficiency. Despite these advancements, challenges remained in further reducing complexity for highly constrained environments. Aboud et al. (2008) advanced RSA by extending its operations beyond integers to incorporate square matrices. This modification aimed to increase flexibility in encryption. However, adapting traditional RSA operations to these more general mathematical structures raised questions about the practical implementation across various platforms [6]. In efforts to improve speed and security, Pradhan and Sharma [3] presented a hybrid RSA cryptosystem utilizing Batch RSA and the MPrime method. By employing multiple prime numbers alongside the Chinese Remainder Theorem (CRT), the authors demonstrated significant improvements in decryption speed without compromising security. However, their work underscored the necessity of balancing accelerated decryption with high security levels in environments requiring frequent decryption. With advancements in computational power, the security of RSA faced new vulnerabilities. Kapoor (2013) introduced a modified RSA algorithm employing four prime numbers and multiple public keys, enhancing the complexity of the modulus. While this approach improved security, it also led to slower encryption, necessitating further exploration into optimizing this balance for security-critical applications [7]. Obaid (2020) emphasized the importance of selecting larger primes for RSA performance, particularly when transmitting sensitive information over insecure networks. By utilizing MATLAB to handle large prime numbers, the study indicated that the selection of large primes significantly enhanced security. Nonetheless, it identified a gap in adapting RSA to modern platforms, such as mobile devices and the Internet of Things (IoT), where efficiency and scalability are paramount [8]. In a more recent study, Jain et al. (2020) explored the security benefits of increasing the number of prime numbers in RSA. Their findings illustrated that higher numbers of primes make factorization significantly more difficult, thereby improving security. However, the trade-off of increased key generation and encryption times highlighted concerns regarding the practicality of such implementations in real-world applications [9]. The authors in [10] proposed a double encryption mechanism utilizing multiple primes, significantly enhancing security. While this approach provided robust protection, it resulted in slower key generation and encryption processes, emphasizing the ongoing trade-off between enhanced security and computational efficiency. Hermawan et al. (2021) focused on modifying RSA for resource-constrained environments like single-board computers (SBCs). Their enhanced RSA method combined eight primes with the CRT, yielding improvements in key generation and encryption speeds—over 21 times faster than traditional RSA. However, the study recognized the need for further reduction of resource consumption while maintaining high security [11]. Ivanov and Stoianov [12] expanded the security discourse by analyzing the arithmetic properties of primes. Their research suggested that the choice of primes could compromise RSA security, highlighting the necessity for optimized selection standards that balance efficiency with security in key generation.

III. EXPERIMENTAL SETUP

Table I provides an overview of the simulation setup utilized for the comparison between the proposed RSA algorithm and the traditional RSA algorithm.

Table I. Simulation setup

Processor	10th Gen Intel Core i7
Memory	16GB RAM
Programming Language	Python
Benchmark	Traditional / Standard RSA

The traditional RSA relies on basic random number generation and primality testing, while the proposed RSA incorporates prime sieving, early termination techniques, and cache reuse to enhance performance. Key parameters such as key generation time, encryption time, decryption time, and memory consumption are measured for each algorithm using key bit sizes of 1024, 2048, and 4096, allowing a comprehensive comparison of their efficiency. For the evaluation of algorithm performance, a dataset consisting of 50 randomly generated numbers in the range of 1000 to 99999999999 was utilized. The random numbers were generated using an online number generator. The experiment was repeated 50 times for each algorithm to ensure consistency and statistical significance. During each iteration, the key exchange times, encryption times, decryption times (measured in milliseconds), and

memory consumption (measured in kilobytes) were recorded. The average values for each metric were computed to provide a comprehensive analysis of the algorithms' performance. These results, including both timing and memory usage metrics, are presented in detail in the corresponding tables, offering a thorough comparison of computational efficiency and resource consumption.

IV. PROPOSED OPTIMIZED ALGORITHM

The proposed RSA algorithm introduces several optimizations aimed at enhancing both the performance and efficiency of traditional RSA. The core optimization involves the use of prime sieving techniques, such as the Sieve of Eratosthenes, to precompute small prime numbers. This significantly reduces the computational effort required during the primality testing phase by allowing the system to eliminate non-prime candidates early on. This approach is supported by research on prime sieving methods, which are known to improve the efficiency of prime generation in cryptographic algorithms. Additionally, the proposed algorithm leverages the Miller-Rabin primality test with early termination to improve the accuracy and speed of large prime number verification. This technique ensures that fewer rounds of testing are needed while maintaining the probabilistic guarantees of primality, as suggested by various studies on its efficiency in cryptographic contexts. A further optimization involves the caching of intermediate results. By storing failed candidates, the algorithm avoids redundant computations for previously tested numbers, thus enhancing both time efficiency and resource utilization. The combination of prime sieving, efficient primality testing, and result caching not only reduces the time required for key generation but also enhances the overall memory efficiency. These improvements align with modern research advocating for the integration of caching mechanisms in cryptographic systems to optimize performance. Fig. 1 illustrates the flowchart detailing the steps involved in the key generation, encryption, and decryption processes of the proposed algorithm.

Proposed RSA Pseudocode

Prime Generation

- **Step 1**: Set sieve limit based on bit length
  limit = set_limit(bits)
- **Step 2**: Generate small primes using sieve
  small_primes = sieve(limit)
- **Step 3**: Initialize cache
  cache = {}

Prime Candidate Loop:

- **Step 4**: Generate random odd number
  cand = rand_odd(bits)
- **Step 5**: Eliminate candidates divisible by small primes
  if all(cand % p != 0 for p in small_primes):
- **Step 6**: Perform Miller-Rabin test
  if all(not miller_rabin(cand, a) for a in range(10)):
- **Step 7**: Check cache for duplicate primes
  if cand not in cache:
- **Step 8**: Add candidate to cache
  cache[cand] = True
- **Step 9**: Return the prime candidate
  return cand

Define sieve function

- **Step 10**: Initialize sieve array
  sieve = [True] * (limit + 1)
- **Step 11**: Loop through small primes
  for i in range(2, int(limit ** 0.5) + 1):
- **Step 12**: Mark multiples as non-prime
  if sieve[i]:
  for j in range(i * i, limit + 1, i):
  sieve[j] = False
- **Step 13**: Return prime numbers
  return [p for p, is_prime in enumerate(sieve) if is_prime]

RSA Key Generation

- **Step 14**: Generate first prime p
  p = gen_prime(bits)
- **Step 15**: Generate second prime q
  q = gen_prime(bits)
- **Step 16**: Compute n = p * q
- **Step 17**: Compute φ(n)
  phi_n = (p - 1) * (q - 1)
- **Step 18**: Set public exponent e
  e = 65537
- **Step 19**: Compute modular inverse d
  d = pow(e, -1, phi_n)
- **Step 20**: Return public and private keys
  return (e, n), (d, n)

Encryption & Decryption

RSA Encryption

- **Step 21**: Encrypt message:
  def encrypt (msg, pub_key), return pow(msg, e, n)

RSA CRT Decryption

- **Step 22**: Factor n into p and q
  p, q = factorize(n)
- **Step 23**: Compute dp, dq, and q_inv:
  dp = d % (p - 1), dq = d % (q - 1), q_inv = pow(q, -1, p)
- **Step 24**: Decrypt with p and q:
  m1 = pow(cipher, dp, p), m2 = pow(cipher, dq, q)
- **Step 25**: Combine results using
  h = (q_inv * (m1 - m2)) % p
- **Step 26**: Return decrypted message
  return (m2 + h * q) % n

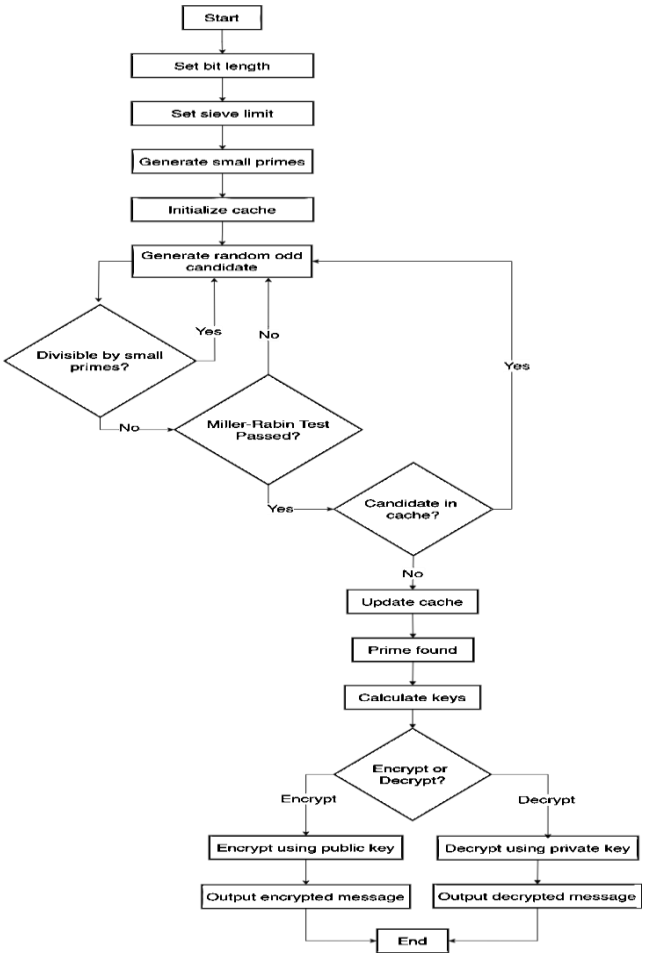

Fig 1: Proposed RSA Algorithm.

V. RESULTS AND DISCUSSION

The algorithms were compared based on the following metrics: key exchange times, encryption and decryption times, and memory consumption between the standard RSA and the Proposed RSA algorithms.

A. Key Exchange Times

Tables II displays a comparison of the key exchange times in milliseconds for both the traditional RSA and Optimized RSA Algorithms.

Table II. Key Exchange Times

ALGORITHM	Key Size (bits)	Average Key Generation Time (ms)
RSA	1024	350
	2048	1800
	4096	25150
Proposed RSA	1024	325
	2048	1675
	4096	22140

The table II presents the average key generation times for both the standard RSA and the optimized RSA algorithms across three key sizes (1024 bits, 2048 bits, and 4096 bits). For the standard RSA, the key generation time increases significantly with the key size, starting at 350ms for 1024 bits, rising to 1800ms for 2048 bits, and reaching 25150ms for 4096 bits. This sharp increase in time highlights the computational cost of generating larger key sizes in the standard RSA algorithm. The optimized RSA algorithm, however, shows a more efficient key generation process across all key sizes. For 1024-bit keys, the optimized RSA generates keys in 325ms, slightly faster than the standard RSA. As the key size increases, the performance gap widens, with the optimized RSA generating 2048-bit keys in 1675ms and 4096-bit keys in 22140ms. This represents a significant reduction in key generation time for larger keys compared to standard RSA, particularly at the 4096-bit level, where the optimized RSA generates keys approximately 12% faster. Both algorithms exhibit longer key generation times as the key size increases, the optimized RSA consistently outperforms the standard RSA, particularly for larger key sizes, offering a more efficient solution without compromising security. This makes the optimized RSA better suited for applications where key generation speed is critical, especially when working with large key sizes.

B. Encryption and Decryption times

Tables III and IV display a comparison of the encryption and decryption times in milliseconds for both the traditional RSA and proposed RSA algorithms.

Table III. Encryption Time

ALGORITHM	Key Size (bits)	Average Encryption Time (ms)
RSA	1024	0.10
	2048	0.35
	4096	0.70
Proposed RSA	1024	0.08
	2048	0.25
	4096	0.50

The table III shows the average encryption times for the standard RSA and the proposed RSA algorithms across three key sizes (1024 bits, 2048 bits, and 4096 bits). For 1024-bit keys, standard RSA records an encryption time of 0.10ms, while the proposed RSA improves slightly, performing at 0.08ms. At this smaller key size, the performance difference is minimal.

As the key sizes increase, the proposed RSA begins to demonstrate clear advantages. For 2048-bit keys, standard RSA takes 0.35 ms, compared to the optimized RSA's 0.25ms, reflecting a significant improvement in encryption speed. For 4096-bit keys, standard RSA shows an encryption time of 0.70ms, while the proposed RSA completes encryption in 0.50ms. This highlights a noticeable gain in efficiency, especially with larger key sizes. While encryption times for both algorithms remain minimal, the proposed RSA consistently outperforms the standard RSA at higher key sizes,

making it a more scalable solution for applications that require faster encryption without sacrificing security.

Table IV. Decryption Times

ALGORITHM	Key Size (bits)	Average Decryption Time (ms)
RSA	1024	0.83
	2048	2.92
	4096	6.28
Proposed RSA	1024	0.32
	2048	1.60
	4096	2.55

The table IV presents the average decryption times for both the standard RSA and the proposed RSA algorithms across three key sizes (1024 bits, 2048 bits, and 4096 bits). For the 1024-bit key size, the standard RSA exhibits a decryption time of 0.83ms, while the proposed RSA reduces this time to 0.32ms. As the key size increases to 2048 bits, the standard RSA shows a decryption time of 2.92ms, whereas the proposed RSA improves this performance further with a decryption time of 1.60ms. At the 4096-bit level, the standard RSA takes 6.28ms, while the proposed RSA completes the decryption in 2.55ms.

The proposed RSA consistently outperforms the standard RSA across all key sizes, showcasing a more efficient decryption process.

C. Memory Consumption

Memory consumption for both the standard RSA and proposed RSA algorithms was measured in kilobyte (KB) using the memory-profiler Python tool, which monitors the amount of RAM utilized during key generation, encryption, and decryption processes for varying key sizes (1024 bits, 2048 bits, and 4096 bits).

Table V. Memory Consumption

ALGORITHM	Key Size (bits)	Average Memory Consumption (KB)
RSA	1024	18.2540
	2048	20.4730
	4096	22.6170
Proposed RSA	1024	16.7890
	2048	17.9320
	4096	19.0860

The table V presents the average memory consumption for both the standard RSA and the proposed RSA algorithms across three key sizes: 1024 bits, 2048 bits, and 4096 bits. For the 1024-bit key size, standard RSA requires 18.2540KB of memory, while the proposed RSA uses slightly less at 16.7890KB, indicating a notable improvement in memory efficiency. As the key sizes increase, the difference in memory usage remains significant. At 2048 bits, standard RSA consumes 20.4730 KB, whereas the proposed RSA shows improved efficiency with 17.9320KB. For the 4096-bit key size, standard RSA records an average of 22.6170KB, while the proposed RSA achieves a more efficient consumption of 19.0860KB. The proposed RSA algorithm demonstrates better memory efficiency across all key sizes, consistently consuming less memory than standard RSA.

VI. CONCLUSION

The study introduced an optimized RSA algorithm aimed at enhancing the efficiency of key generation, encryption, and decryption processes. The improvements were achieved through a combination of prime sieving techniques, cache reuse, and early termination in the Miller-Rabin primality test. These optimizations led to faster cryptographic operations while reducing memory consumption. By integrating these enhancements, the proposed RSA algorithm demonstrated quicker key generation, faster encryption and decryption times, and lower memory usage compared to the traditional RSA algorithm, making it a more efficient and resource-conscious alternative.

REFERENCES

[1] M. R. Khan *et al.*, “Analysis of Elliptic Curve Cryptography & RSA,” *J. ICT Stand.*, vol. 11, no. 4, pp. 355–378, 2023, doi: 10.13052/jicts2245-800X.1142.

[2] A. Joshi and V. Anand, “Design of Novel Key Generation Technique Based RSA Algorithm for Efficient Data Encryption and Decryption,” *ECS Trans.*, vol. 107, no. 1, pp. 2585–2592, 2022, doi: 10.1149/10701.2585ecst.

[3] S. Pradhan and B. K. Sharma, “An Efficient RSA Cryptosystem with BM-PRIME Method,” *Int. J. Inf. Netw. Secur.*, vol. 2, no. 1, pp. 103–108, 2012, doi: 10.11591/ijins.v2i1.1718.

[4] U. M. Maurer, “Fast generation of prime numbers and secure public-key cryptographic parameters,” *J. Cryptol. J. Int. Assoc. Cryptologie Res.*, vol. 8, no. 3, pp. 123–155, 1995, doi: 10.1007/BF00202269.

[5] M. Joye, P. Paillier, and S. Vaudenay, “Efficient generation of prime numbers,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1965 LNCS, pp. 340–354, 2000, doi: 10.1007/3-540-44499-8_27.

[6] S. J. Aboud, M. A. Al-Fayoumi, M. Al-Fayoumi, and H. S. Jabbar, “An efficient RSA public key encryption scheme,” *Proc. - Int. Conf. Inf. Technol. New Gener. ITNG 2008*, pp. 127–130, 2008, doi: 10.1109/ITNG.2008.199.

[7] V. Kapoor, “Data Encryption and Decryption using Modified RSA Cryptography Based on Multiple Public Keys and ‘n’ Prime Number,”

Int. J. Eng. Sci. Res. Technol., vol. 1, no. 2, pp. 713–720, 2013.

[8] T. S. Obaid, "Study A Public Key in RSA Algorithm," *Eur. J. Eng. Res. Sci.*, vol. 5, no. 4, pp. 395–398, 2020, doi: 10.24018/ejers.2020.5.4.1843.

[9] N. Jain, S. S. Chauhan, and A. Raj, "Security Enhancement of RSA Algorithm using Increased Prime Number Set," *Int. J. Eng. Adv. Technol.*, vol. 9, no. 3, pp. 4235–4240, 2020, doi: 10.35940/ijeat.c6278.029320.

[10] M. A. Islam, M. A. Islam, N. Islam, and B. Shabnam, "A Modified and Secured RSA Public Key Cryptosystem Based on 'n' Prime Numbers," *J. Comput. Commun.*, vol. 06, no. 03, pp. 78–90, 2018, doi: 10.4236/jcc.2018.63006.

[11] N. T. E. Hermawan, E. Winarko, and A. Ashari, "Eight Prime Numbers of Modified RSA Algorithm Method for More Secure Single Board Computer Implementation," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 11, no. 6, pp. 2375–2384, 2021, doi: 10.18517/ijaseit.11.6.13700.

[12] A. Ivanov and N. Stoianov, "Implications of the Arithmetic Ratio of Prime Numbers for RSA Security," *Int. J. Appl. Math. Comput. Sci.*, vol. 33, no. 1, pp. 57–70, 2023, doi: 10.34768/amcs-2023-0005.