Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Security and Efficiency Evaluation of RSA Algorithm with Multiple Key Sizes Using Python and Web-Based Platforms

Sonam Gawande

sonagawande7@gmail.com

Mahatma Gandhi Vidyamandir's Maharaja Sayajirao Gaikwad Arts, Science & Commerce College Tal. Malegaon, Dist. Nashik

Kailas Ahire

Mahatma Gandhi Vidyamandir's Maharaja Sayajirao Gaikwad Arts, Science & Commerce College Tal. Malegaon, Dist. Nashik

Research Article

Keywords: Public key cryptography, RSA, Encryption, Decryption, Python and web tools

Posted Date: March 3rd, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9006567/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Security and Efficiency Evaluation of RSA Algorithm with Multiple Key Sizes Using Python and Web-Based Platforms

Sonam Namdev Gawande[1] Kailas Sahadu Ahire[2]

[1]Mahatma Gandhi Vidyamandir's,
Maharaja Sayajirao Gaikwad Arts, Science & Commerce College, Malegaon,
Malegaon Camp, District Nashik, Maharashtra, India.
E-mail: sonamgawande@gmail.com
[2]Mahatma Gandhi Vidyamandir's,
Maharaja Sayajirao Gaikwad Arts, Science & Commerce College, Malegaon,
Malegaon Camp, District Nashik, Maharashtra, India.
E-mail: ksahire111@gmail.com

1 Abstract

This paper offers a comprehensive analysis of the RSA public key cryptosystem, focusing on its security and efficiency. The size of the key used determines how the RSA algorithm is evaluated. In this study, a Python implementation of the RSA algorithm with varying key sizes is presented, and the performance in terms of key generation, encryption, and decryption time is examined. Tables and graphical representations are used to display the results.The processes of encryption, decoding, key creation, and mathematical principles are covered. System specs are used to compare various RSA keys. Various key sizes and pre-made graphs for encryption time, decryption time, and security strength are used to assess performance. The use of common cryptographic libraries for Python implementation is also covered.In order to create a more secure RSA algorithm, this research study attempts to modify the RSA algorithm.RSA cryptosystem algorithm is widely used cryptographic algorithm in network security but it has problem of integer's factorisation for small numbers. This study presents and compares a number of RSA cryptosystem improvements that aim to increase security and expedite the encryption and decryption process.

2 Keywords

Public key cryptography, RSA, , Encryption, Decryption,Python and web tools

3 Introduction

A cryptographer uses cryptography, which is the science or art of communication security.[1] Cryptanalysis is the science and art of deciphering ciphertext into plaintext The process is known as cryptanalysis. The Greek terms "kryptos," which means to conceal, and "graphein," which means to write, are the origin of the word cryptography. Cryptography Algorithm always consists of two parts, namely encryption, and decryption. The process of converting a readable communication into an unreadable one (ciphertext) is called encryption. Unread messages are returned to unread messages during decryption, which is the reverse of encryption. One or more cryptographic keys control the encryption and decryption process. Cryptosystems are systems that include all potential plaintext, ciphertext, and keys along with a cryptographic algorithm.In contemporary digital communication, cryptography is essential to information security.There are two types of cryptography symmetric and asymmetric[2]

Symmetric key cryptography: A shared secret key between the sender and the recipient enables this process. As a result, if a third party obtains this key, the communication may be jeopardized. Since both communication parties have the shared secret, symmetric key cryptography uses the same common key for both encryption and decryption.
Despite being quicker and more computationally efficient than public-key techniques, symmetric key encryption has a number of serious problems, most notably the difficulty of distributing and managing safe keys across numerous users. The length of the key and the safe upkeep and sharing of the key across all parties play a major role in the overall strength of symmetric encryption.

Asymmetric key cryptography: Asymmetric cryptography requires two distinct keys: one for encryption and another for decryption. The encryption key, known as the public key, is openly distributed, whereas the decryption key, referred to as the private key, is kept confidential. Consequently, any entity with access to the public key can encrypt data, but only the authorized user or device possessing the private key can decrypt it. The matching private key must stay with the intended recipient alone, even if the public key is shared by all users or devices taking part in a communication session. Secure key exchange is made possible by asymmetric cryptographic algorithms, despite the fact that they are computationally slower than conventional symmetric algorithms and may be more vulnerable to specific attacks. Therefore, combining symmetric and asymmetric

cryptographic techniques to take use of each method's advantages results in the best encryption strategy.
RSA is one of the most commonly used asymmetric cryptographic algorithms.[3] Its security relies on the difficulty of factoring large composite numbers. RSA is frequently used in secure communication, digital signatures, and key exchange protocols.Data security and confidentiality issues are an important aspect of information systems. Because of the confidentiality and security of data, it is essential always to be concerned both for common security, as well as for the privacy of individuals.[4] It was discovered by Ron Rivest, Adi Shamir, and Leonard Adleman in 1978. RSA stands for the initials of the names of the three of them. [5] [6]

4 RSA Algorithm

There are three main processes in the RSA algorithm:[7]

1. Key Generation
2. Encryption
3. Decryption

A. Key Generation

- Generate two large random prime numbers p and q such that $\gcd(p, q) = 1$.
- Compute
$$n = p \times q$$
- Compute Euler's totient function
$$\phi(n) = (p-1)(q-1)$$
- Choose a public key integer e such that
$$1 < e < \phi(n) \quad \text{and} \quad \gcd(e, \phi(n)) = 1$$
- Compute the private key integer d such that
$$d \equiv e^{-1} \pmod{\phi(n)}$$
- The public key is (e, n) and the private key is (d, n).

B. Encryption

- Let the public key be (e, n), the message be m, and the cipher-text be c.
- Encrypt the message using the public key:

$$c = m^e \bmod n$$

- Sender A sends the cipher-text c to receiver B.

C. Decryption

- The receiver uses the private key (d, n).
- Decrypt the cipher-text to obtain the original message:

$$m = c^d \bmod n$$

- Receiver B reads the original message.

5 Key Generation Flowchart

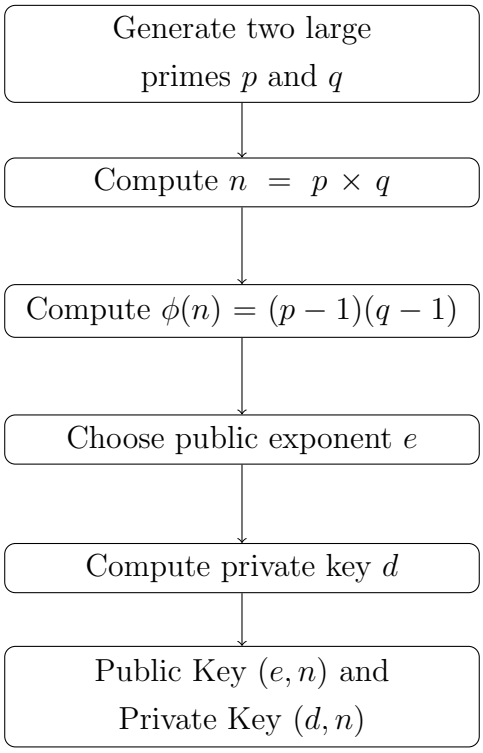


Figure 1: RSA Key Generation Flowchart

6 Python Implementation

Libraries Used

- `pycryptodome`
- `matplotlib`
- `numpy`
- `time`

Tools and Software Used

Sr. No.	Component	Description
1	Programming Language	Python 3.x
2	Libraries Used	cryptography, matplotlib, time
3	Integrated Development Environment (IDE)	Google colab
4	Operating System	Windows

Table 1: Tools and Software Used for Implementation

RSA Algorithm

1. Generate RSA public and private keys
2. Convert plaintext message into bytes
3. Encrypt the message using the public key
4. Decrypt the ciphertext using the private key
5. Measure execution time for each operation

Python Implementation[8]

```
from cryptography.hazmat.primitives.asymmetric import rsa,
   padding
from cryptography.hazmat.primitives import hashes
import time
import matplotlib.pyplot as plt

message ="MATHEMATICS␣RSA␣PROJECT"
```

```
key_sizes = [512, 1024, 2048]
keygen_time = []
encrypt_time = []
decrypt_time = []

for size in key_sizes:
    start = time.time()
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=size
    )
    public_key = private_key.public_key()
    keygen_time.append((time.time() - start) * 1e6)

    start = time.time()
    ciphertext = public_key.encrypt(
        message,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    encrypt_time.append((time.time() - start) * 1e6)

    start = time.time()
    plaintext = private_key.decrypt(
        ciphertext,
        padding.OAEP(
            mgf=padding.MGF1(algorithm=hashes.SHA256()),
            algorithm=hashes.SHA256(),
            label=None
        )
    )
    decrypt_time.append((time.time() - start) * 1e6)
```

7 Graphical Representation

The following graphs show the effect of key size on RSA performance.

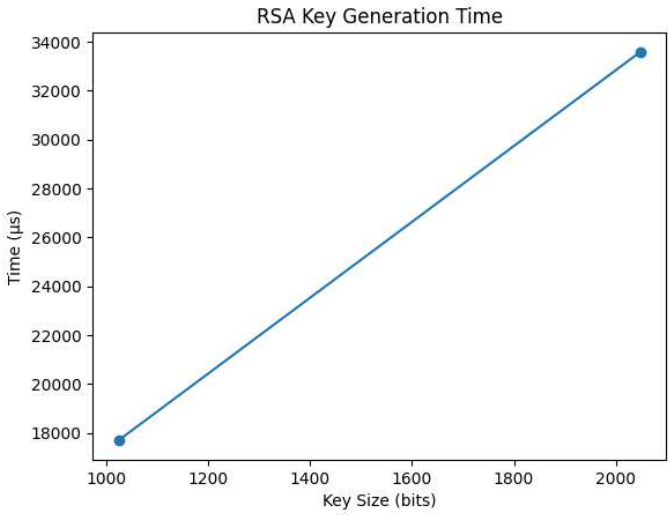


Figure 2: Key Generation Time vs Key Size

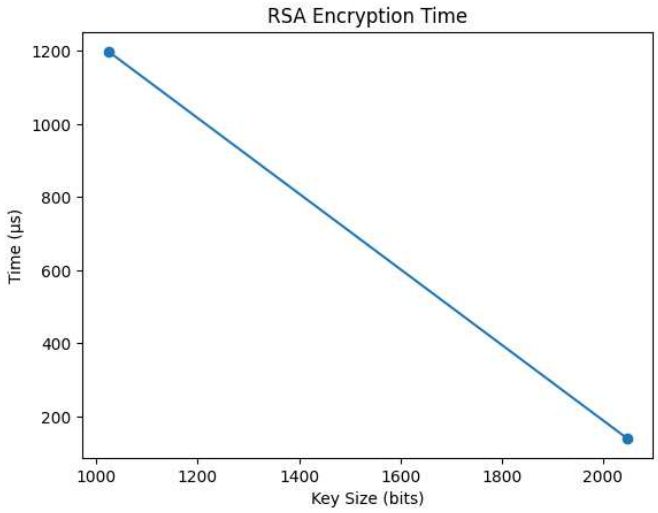


Figure 3: Encryption Time vs Key Size

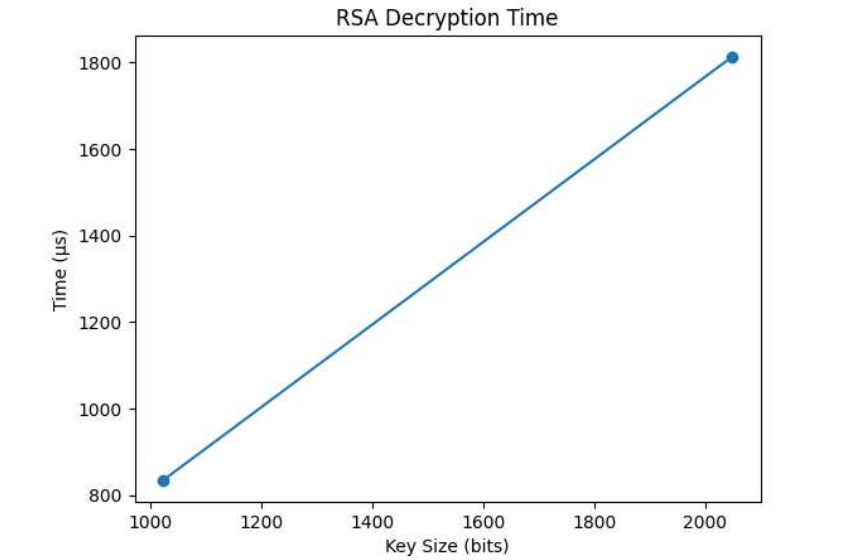


Figure 4: Decryption Time vs Key Size

Observation and Results

Key Size (bits)	**Key Generation (μs)**	**Encryption (μs)**	**Decryption (μs)**
512	Low	Very Low	Low
1024	Medium	Medium	Medium
2048	High	High	Very High

Table 2: Performance Comparison of RSA with Different Key Sizes

8 Additional Implementation Using Online RSA Tool

In addition to the Python-based implementation, the encryption and decryption of RSA were also experimentally verified using an online cryptographic tool available at `https://www.devglan.com/online-tools/rsa-encryption-decryption`. This approach provides a practical, code-free environment to observe RSA behavior with different key sizes and validate theoretical concepts.

8.1 Description of the Online RSA Tool

The Devglan RSA tool is a web-based application that allows users to:

- Generate RSA public and private key pairs
- Select different RSA key sizes

- Encrypt plaintext using the public key
- Decrypt ciphertext using the private key

The tool generates the public key in **X.509 format** and the private key in **PKCS#8 format**, which are widely used in modern cryptographic applications such as SSL/TLS and digital certificates.

8.2 RSA Key Sizes Used

In this project, the following RSA key sizes were selected to study their impact on security and performance:

- 512-bit RSA
- 1024-bit RSA
- 2048-bit RSA

These key sizes allow a comparative study between insecure, deprecated, and currently recommended RSA configurations.

8.3 Theoretical Analysis of RSA Key Sizes

512-bit RSA A 512-bit RSA key uses relatively small prime numbers, making the modulus $n = pq$ vulnerable to factorization attacks. As of 2025, such keys can be broken using modern computing resources and are considered insecure. Hence, 512-bit RSA is used only for academic and demonstration purposes.

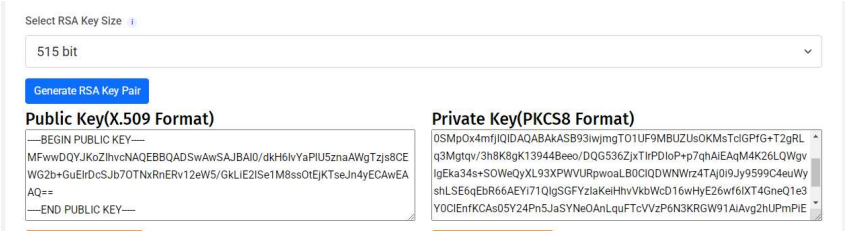


Figure 5: RSA Key Pair with key size 512

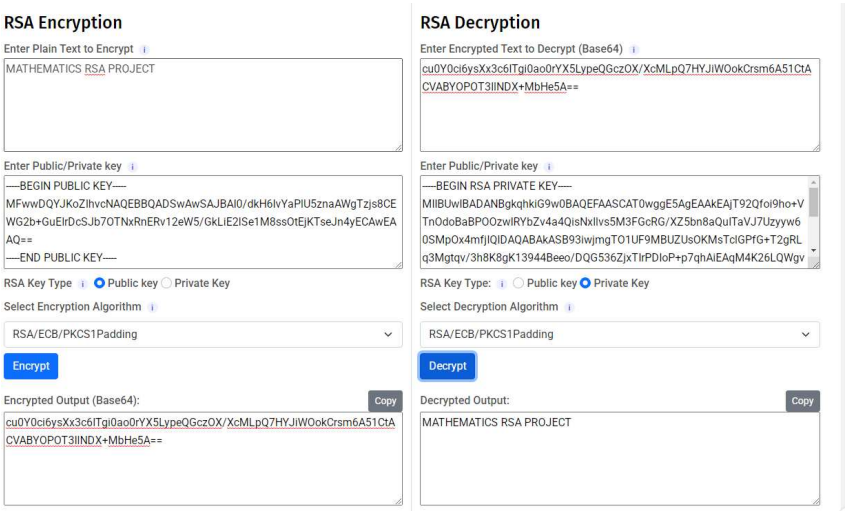


Figure 6: Encryption and decryption using key size 512

1024-bit RSA The 1024-bit RSA key provides higher security than 512-bit keys but is no longer recommended for long-term security. Advances in computational power and cryptanalysis have rendered this key size insufficient against sophisticated adversaries.

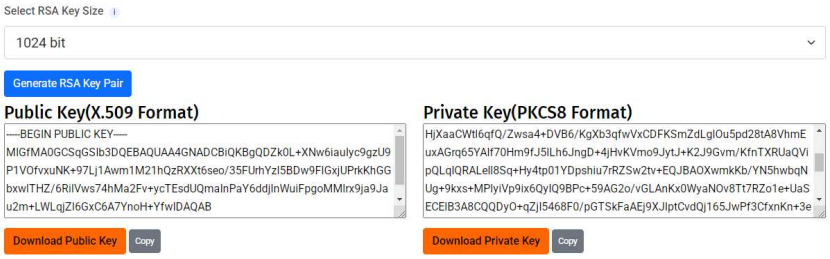


Figure 7: RSA Key Pair with key size 1024

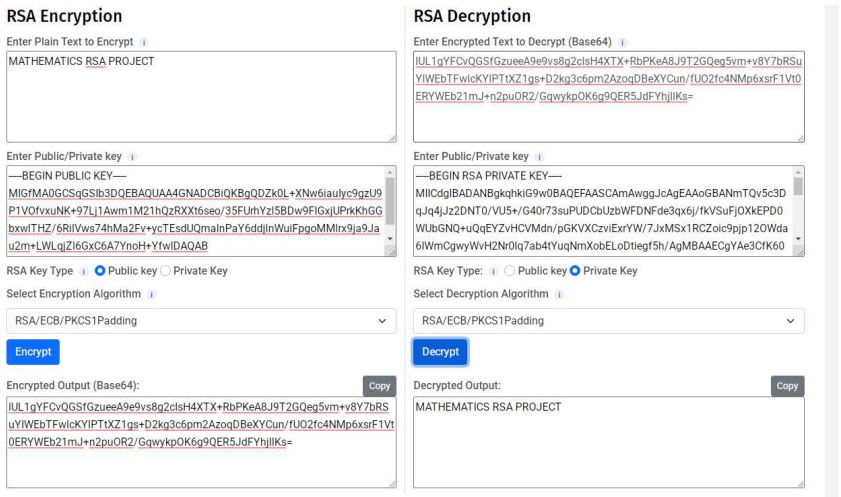


Figure 8: Encryption and decryption using key size 1024

2048-bit RSA RSA with a 2048-bit key uses very large prime numbers, making factorization computationally infeasible using classical computers. This key size is widely accepted as secure and is recommended for most practical applications in current cryptographic standards.

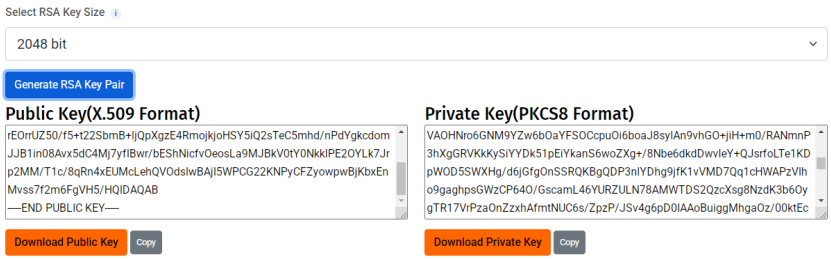


Figure 9: RSA Key Pair with key size 2048

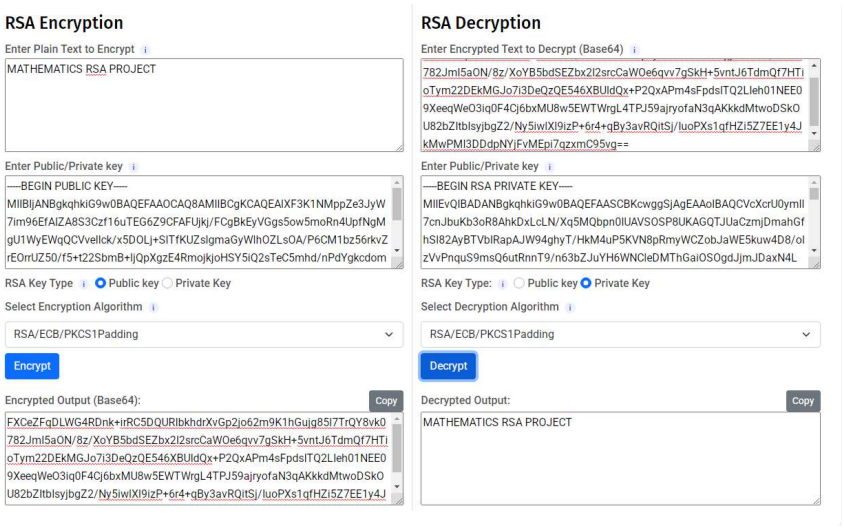


Figure 10: Encryption and decryption using key size 2048

8.4 Observation Using Online RSA Tool

Key Size in bit	Key Generation	Encryption	Decryption	Security Level
512	Very Fast	Very Fast	Very Fast	Very Low
1024	Fast	Fast	Medium	Medium
2048	Slow	Medium	Slow	High

Table 3: Performance Observation Using Online RSA Tool

8.5 Comparison Between Python Implementation and Online Tool

Aspect	**Python RSA Implementation**	**Online RSA Tool**
Programming Required	Yes	No
Ease of Use	Moderate	Very Easy
Mathematical Control	High	Medium
Execution Visibility	High	Limited
Educational Value	High	High

Table 4: Comparison of RSA Implementations

9 Conclusion

RSA is still a popular public-key cryptography algorithm that guarantees safe communication. Since sensitive information must only be accessible to the intended recipient, data security and confidentiality are essential criteria. A system that can effectively maintain robust data confidentiality can be constructed by using the RSA algorithm. Technically speaking, RSA provides a conceptually straightforward encryption method; yet, the usage of huge prime numbers makes it computationally impossible to extract the original plaintext from the ciphertext without the private key.Key size, the choice of large and secure prime numbers (p) and (q), and potential encryption flaws are all important considerations when creating public and private keys. The computational challenge of factoring huge composite numbers into their prime factors is the underlying basis of RSA's security. RSA has a low computing speed when compared to symmetric encryption techniques, yet offering a great degree of security.

References

[1] Dian Kurnia, Haida Dafitri, Sugianto, Mardiana, and Andysah Putera Utama Siahaan. Rsa 32-bit implementation technique. *International Journal of Recent Trends in Engineering & Research*.

[2] Omar Salah, Ahmed El-Sawy, and Mohamed Taha. A hybrid algorithm for enhancement of the data security during network transmission based on rsa and dh. *International Journal of Intelligent Engineering and Systems*, 16(3):614–???, 2023.

[3] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[4] Dian Kurnia, Haida Dafitri, Sugianto, Mardiana, and Andysah Putera Utama Siahaan. Rsa 32-bit implementation technique. *International Journal of Computer Science and Information Technology*, 2018. Indonesia.

[5] A. Gayathri, K. Ravisagar, and E. Pragnavi. Implementation of rsa with digital signatures using four prime numbers. *Indian Journal of Scientific Research*, 17(2):99–101, 2018. ISSN (Online): 2250-0138.

[6] Devi V. Kumar. A hybrid security protocol using python. *International Journal of Computer Science and Information Technology Research*, 2(4):9–16, 2014. ISSN (Online): 2348-120X.

[7] Abhishek and V. Vandana. A study on modified rsa algorithm in network security. *International Research Journal of Modernization in Engineering Technology and Science (IRJMETS)*, 4(4), April 2022. Open Access Journal.

[8] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 7th edition, 2017.