Journal of
Data Acquisition and Processing

ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.777495

# COMPARING THE RESILIENCE AND EFFICIENCY OF POST-QUANTUM AND ASYMMETRIC ENCRYPTION

**Gokul Gopakumar, Vishnu Anilkumar Nair, Dr. Jayashree J***
School of Computer Science and Engineering, Vellore Institute of Technology, Vellore, India

**Abstract** : With the advent of quantum computing, traditional encryption methods become more vulnerable to attacks that exploit their computational weaknesses. In response, NIST launched a competition for post-quantum cryptography algorithms that can withstand such attacks. One of the promising entrants is CRYSTALS-Kyber, a post-quantum cryptographic algorithm that may be secure from quantum attacks. However, the effectiveness of these algorithms has yet to be demonstrated in real-world applications. This paper presents a comparative analysis of three encryption schemes: Kyber, McEliece, and RSA (public key encryption). We evaluate these algorithms based on key parameters such as memory consumption, encryption time, decryption time, and key generation time. By comparing and contrasting these algorithms, our study aims to assess the potential utility and applicability of post-quantum cryptography in practical contexts.

**Keywords :** CRYSTALS-Kyber, decryption time, encryption schemes, encryption time, key generation time, Kyber, McEliece, memory consumption, NIST, post-quantum cryptography, public key encryption, quantum attacks, RSA, secure, traditional encryption, vulnerabilities.

## 1. Introduction

Classical encryption, also known as conventional encryption, is a type of data protection that uses several methods to turn data into an unreadable format. It encodes and decodes data using a key and is widely used in modern communication and data storage systems. It is effective at preventing unwanted access to data, but it is vulnerable to attacks from more powerful computers and novel hacking techniques. It gets easier to break standard encryption and gain access to sensitive information as computer power improves. As a result, more secure encryption methods that can withstand attacks from strong computer technologies are required. Post-quantum encryption is a type of encryption designed to survive quantum computer attacks. Unlike classical and quantum encryption, which can be exploited by quantum computers, post-quantum encryption employs mathematical techniques that are impervious to quantum algorithms. Post-quantum encryption is a relatively newer field of study, but its importance continues to rise as quantum computer technology progresses. Post-quantum encryption approaches are built on mathematical difficulties that conventional and quantum computers are predicted to find difficult to solve. Lattice-based encryption, code-based cryptography, hash-based cryptography, and multivariate cryptography are among the challenges. Post-quantum encryption algorithms are designed to be compatible with existing communication and data storage systems, allowing for easy integration into existing technology.

Lattice-based encryption is a type of cryptography that relies on the mathematical properties of lattices to provide security. In this method, data is encrypted by encoding it as a point in a high-dimensional lattice. Lattice-based encryption schemes are known for their resistance to quantum attacks, making them a promising area of research in post-quantum cryptography.

Code-based cryptography is a type of encryption that relies on error-correcting codes to provide security. In this method, data is encoded as a sequence of bits using a special code that can correct errors that occur during transmission or storage. Code-based encryption schemes are known for their resistance to quantum attacks, making them a promising area of research in post-quantum cryptography. However, they are relatively slow and have not been widely adopted yet.

## 2. Proposed Methodology

The RSA cryptography system is a popular public-key cryptography system. It derives its security from the challenging nature of factoring the product of two huge prime integers. RSA cryptography encrypts data using a public key and decrypts it with a private key. The public key can be freely transmitted, whilst the owner keeps the private key private. RSA cryptography is frequently used in secure communication, digital signatures, and sensitive data encryption.

Longer key sizes are necessary to maintain the same degree of security as computers become more powerful.

McEliece is a public-key encryption method that, unlike RSA, does not rely on huge integers to be factored. The McEliece system's security is based on the difficulty of solving specific mathematical challenges related to coding theory. The system encrypts using a public key and decrypts with a private key, with the public key obtained from a randomly generated linear code. Because of its resilience to quantum computer attacks the McEliece system is a good contender for post-quantum cryptography.

However, when compared to other public-key encryption systems, the method requires bigger key sizes, which can make it computationally costly and ineffective for various applications.

Kyber is a post-quantum key encapsulation mechanism (KEM) based on the difficulty of solving specific mathematical problems in lattices. The Kyber Key Exchange Module (KEM) is used to securely exchange keys between two parties, where the key is created by the server and given to the client. The method is designed to withstand attacks by quantum computers, which are predicted to be capable of breaking conventional public-key encryption systems. Kyber is seen as a viable contender for post-quantum encryption due to its efficiency, small key sizes, and resilience to both conventional and quantum assaults.

We can compare RSA, McEliece, and Kyber based on a few common and crucial features that will aid in our analysis.

From Fig. 1, we can understand the basic working of asymmetric cryptography.

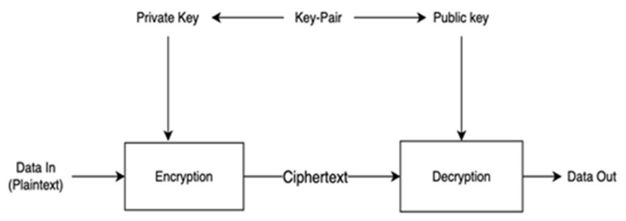


Fig.1. Asymmetric Cryptography

## 2.1 RSA

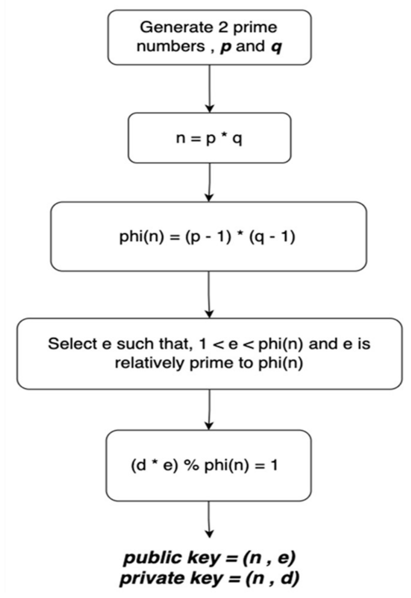

Generate 2 prime numbers , *p* and *q*
n = p * q
phi(n) = (p - 1) * (q - 1)
Select e such that, 1 < e < phi(n) and e is relatively prime to phi(n)
(d * e) % phi(n) = 1
*public key = (n , e)*
*private key = (n , d)*

Fig.2. KeyGen for RSA

Here's a detailed overview of the RSA encryption and decryption processes (from Fig.2 & 3).
Encryption Process:

1) Generate Key Pair :

- Generate a key pair by selecting two prime numbers, p and q, of equal length and calculating the modulus n = p * q.
- Calculate Euler's totient function phi(n) = (p-1) * (q-1).
- Select a public exponent e that is relatively prime to phi(n) and calculate the corresponding private exponent d such that e * d ≡ 1 (mod phi(n)).
- Export the public key as the pair (n, e) and the private key as the pair (n, d).

2) Convert the plaintext to a buffer.

- Convert the plaintext message into a buffer format that can be processed by the RSA algorithm.

3) Encrypt the Plaintext:

- Use the public key (n, e) to encrypt the plaintext buffer using the RSA encryption algorithm.
- The encrypted message is now ready for transmission.

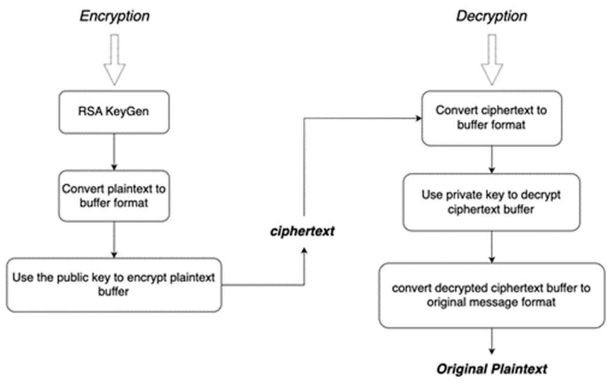

Encryption
RSA KeyGen
Convert plaintext to buffer format
Use the public key to encrypt plaintext buffer
ciphertext
Decryption
Convert ciphertext to buffer format
Use private key to decrypt ciphertext buffer
convert decrypted ciphertext buffer to original message format
Original Plaintext

Fig.3. RSA Algorithm

**Decryption Process:**

1) Convert the Encrypted Message to a Buffer:

- Convert the encrypted message into a buffer format that can be processed by the RSA algorithm.

2) Decrypt the Ciphertext:

- Use the private key (n, d) to decrypt the ciphertext buffer using the RSA decryption algorithm.
- The decrypted message is now in buffer format.

3) Convert the Decrypted Buffer to Plaintext:

- Convert the decrypted message buffer back to the plaintext message format.

4) Return the Plaintext:

- The decrypted plaintext is now ready for use.

## 2.2 McEliece

Here's a high-level overview of the encryption and decryption process (from Fig.4 & 5).

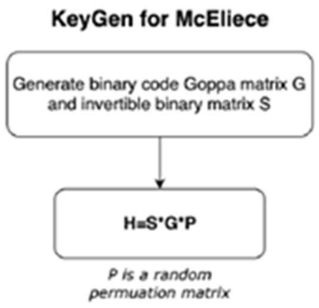

KeyGen for McEliece
Generate binary code Goppa matrix G and invertible binary matrix S
H=S*G*P
P is a random permuation matrix

Fig.4. Keygen for McEliece Cryptography

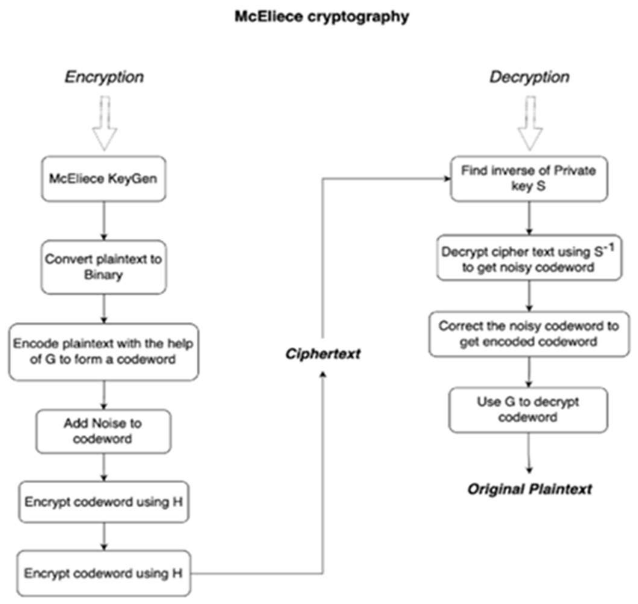

McEliece cryptography
Encryption
McEliece KeyGen
Convert plaintext to Binary
Encode plaintext with the help of G to form a codeword
Add Noise to codeword
Encrypt codeword using H
Encrypt codeword using H
Ciphertext
Decryption
Find inverse of Private key S
Decrypt cipher text using S⁻¹ to get noisy codeword
Correct the noisy codeword to get encoded codeword
Use G to decrypt codeword
Original Plaintext

Fig.5. McEliece Cryptography

**Encryption Process:**

1) Generate Public and Private Keys:
- Randomly generate a binary Goppa code matrix G and an invertible binary matrix S.
- Compute the matrix H = S * G * P, where P is a random permutation matrix.
- Use H as the public key, and keep G, S, and P as the private key.

2) Convert the Plaintext to Binary Format:
- Convert the plaintext message into a binary format, for example, ASCII encoding.

3) Encode the Plaintext:
- Break the plaintext into blocks of appropriate size.
- Encode each block into a binary codeword using the binary Goppa code generator matrix G.

4) Add Noise to the Codeword:
- Add random noise to the encoded codeword.

5) Encrypt the Codeword:
- Multiply the noisy codeword with the public key matrix H to generate the ciphertext.

6) Return the Ciphertext:
- The encrypted message is now ready for transmission.

Decryption Process:

1) Generate the inverse of S:
- Use the private key S to generate the inverse of S.

2) Decrypt the Ciphertext:
- Multiply the ciphertext with the inverse of S to recover the noisy codeword.

3) Correct the Noisy Codeword:
- Use the binary Goppa code syndrome decoder to correct the noisy codeword and get the encoded codeword.

4) Decode the Codeword:

- Use the binary Goppa code generator matrix G to decode the encoded codeword into the original plaintext.

5) Return the Plaintext:

- The decrypted plaintext is now ready for use.

## 2.3 Kyber768

The Kyber768 variant uses a public-key system that relies on the hardness of the learning with errors (LWE) problem in the Ring Learning with Errors (RLWE) setting. Here's a detailed overview of the Kyber 768 encryption and decryption processes. (from Fig.6 & 7).

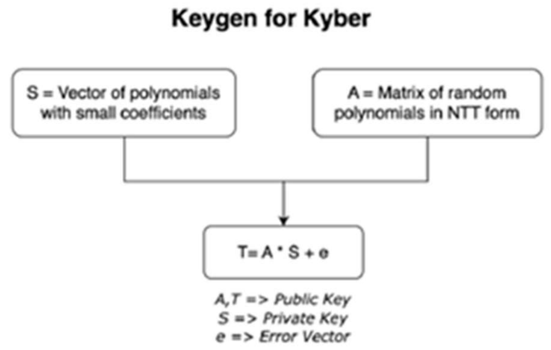

Keygen for Kyber
S = Vector of polynomials with small coefficients
A = Matrix of random polynomials in NTT form
T= A * S + e
A,T => Public Key
S => Private Key
e => Error Vector

Fig.6. Kyber Keygen

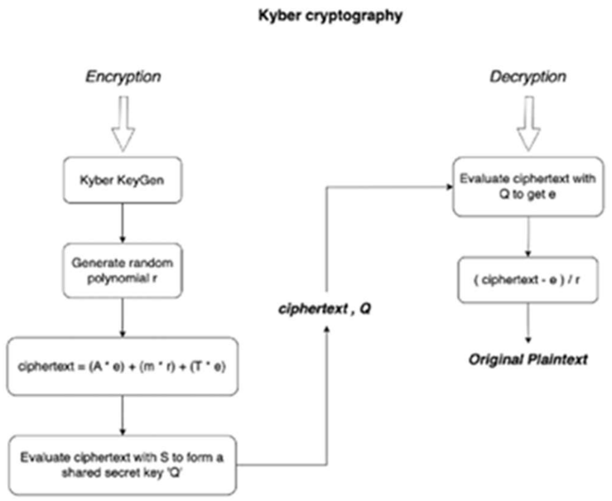

Kyber cryptography
Encryption
Kyber KeyGen
Generate random polynomial r
ciphertext = (A * e) + (m * r) + (T * e)
Evaluate ciphertext with S to form a shared secret key 'Q'
ciphertext , Q
Decryption
Evaluate ciphertext with Q to get e
( ciphertext - e ) / r
Original Plaintext

Fig.7. Kyber Algorithm

**Encryption Process:**

1. Generate Key Pair:

- Generate a secret key s by generating a polynomial with small coefficients from a uniformly random source.
- Compute the public key by evaluating the polynomial on a set of points (a0, a1, ..., a511) generated from a hash of the public key.

2. Generate Shared Secret:

- Generate a random polynomial r with small coefficients from a uniformly random source.
- Generate a ciphertext by adding the public key multiplied by the error e with small coefficients and the message m multiplied by the random polynomial r.

- Generate a shared secret by evaluating the ciphertext polynomial on the secret key s.

3. Return the Ciphertext and Shared Secret:

- The encrypted message and shared secret are now ready for transmission.

Decryption Process:

1) Recover the Error:

- Evaluate the ciphertext polynomial on the secret key s to obtain the error e.

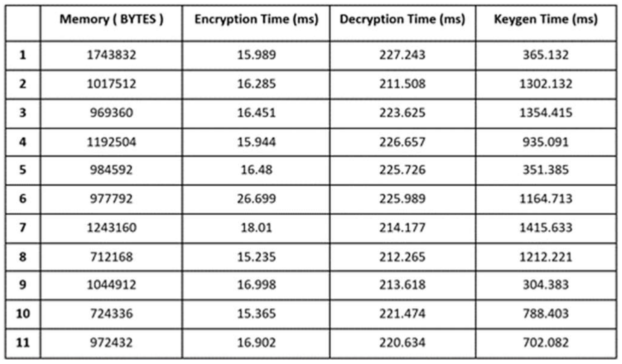
2) Recover the Message:

- Subtract the error from the ciphertext polynomial to obtain the polynomial representing the message multiplied by the random polynomial r.
- Recover the message by dividing the polynomial by r.

3) Return the Plaintext:

- The decrypted plaintext is now ready for use.

## 3. Comparative Analysis

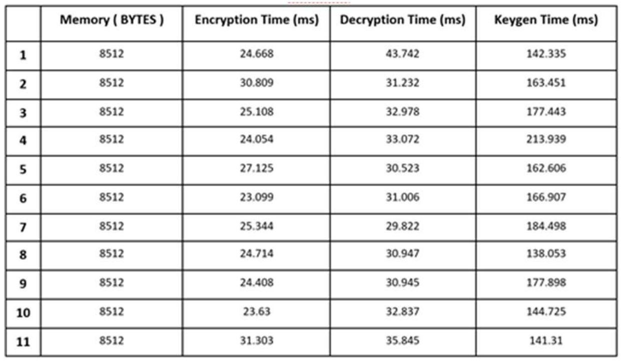


| | Memory ( BYTES ) | Encryption Time (ms) | Decryption Time (ms) | Keygen Time (ms) |
|---|---|---|---|---|
| 1 | 1743832 | 15.989 | 227.243 | 365.132 |
| 2 | 1017512 | 16.285 | 211.508 | 1302.132 |
| 3 | 969360 | 16.451 | 223.625 | 1354.415 |
| 4 | 1192504 | 15.944 | 226.657 | 935.091 |
| 5 | 984592 | 16.48 | 225.726 | 351.385 |
| 6 | 977792 | 26.699 | 225.989 | 1164.713 |
| 7 | 1243160 | 18.01 | 214.177 | 1415.633 |
| 8 | 712168 | 15.235 | 212.265 | 1212.221 |
| 9 | 1044912 | 16.998 | 213.618 | 304.383 |
| 10 | 724336 | 15.365 | 221.474 | 788.403 |
| 11 | 972432 | 16.902 | 220.634 | 702.082 |

Table 1. RSA Metric Analysis

Here's a breakdown of what the columns in the table represent:

- Memory (BYTES): The size of the memory used for RSA encryption (in bytes).
- Encryption Time (ms): The time it takes to encrypt a message using RSA (in milliseconds).
- Decryption Time (ms): The time it takes to decrypt an encrypted message using RSA (in milliseconds).
- Keygen Time (ms): The time it takes to generate an RSA key pair (in milliseconds).



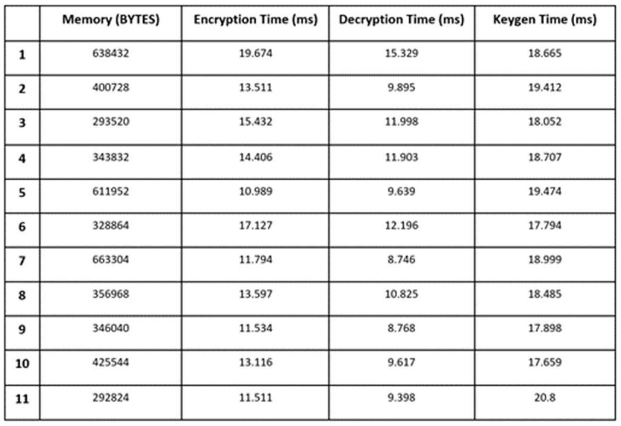
| | Memory ( BYTES ) | Encryption Time (ms) | Decryption Time (ms) | Keygen Time (ms) |
|---|---|---|---|---|
| 1 | 8512 | 24.668 | 43.742 | 142.335 |
| 2 | 8512 | 30.809 | 31.232 | 163.451 |
| 3 | 8512 | 25.108 | 32.978 | 177.443 |
| 4 | 8512 | 24.054 | 33.072 | 213.939 |
| 5 | 8512 | 27.125 | 30.523 | 162.606 |
| 6 | 8512 | 23.099 | 31.006 | 166.907 |
| 7 | 8512 | 25.344 | 29.822 | 184.498 |
| 8 | 8512 | 24.714 | 30.947 | 138.053 |
| 9 | 8512 | 24.408 | 30.945 | 177.898 |
| 10 | 8512 | 23.63 | 32.837 | 144.725 |
| 11 | 8512 | 31.303 | 35.845 | 141.31 |

**Table 2. McEliece Metric Analysis**

The provided data appears to be a table showing the encryption time, decryption time, and key generation time for 11 different instances of the McEliece cryptosystem using the same memory size (8512 bytes).

Here's a breakdown of what the columns in the table represent:

- Memory (BYTES): The size of the memory used for McEliece encryption (in bytes).
- Encryption Time (ms): The time it takes to encrypt a message using McEliece (in milliseconds).
- Decryption Time (ms): The time it takes to decrypt an encrypted message using McEliece (in milliseconds).
- Keygen Time (ms): The time it takes to generate a McEliece key pair (in milliseconds).



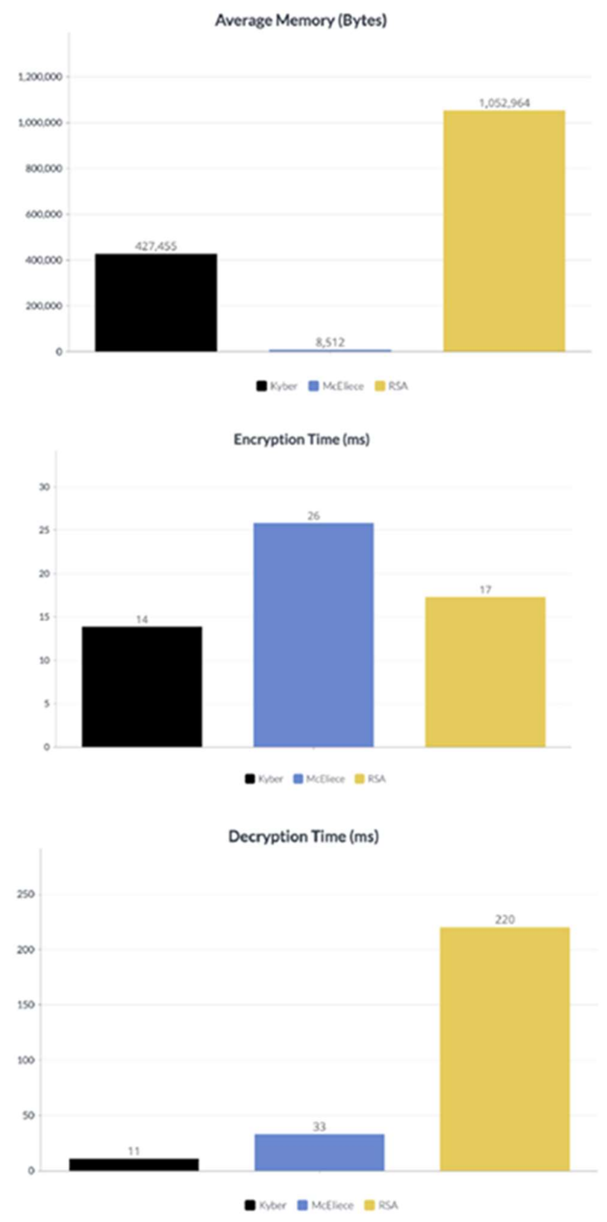
| | Memory (BYTES) | Encryption Time (ms) | Decryption Time (ms) | Keygen Time (ms) |
|---|---|---|---|---|
| 1 | 638432 | 19.674 | 15.329 | 18.665 |
| 2 | 400728 | 13.511 | 9.895 | 19.412 |
| 3 | 293520 | 15.432 | 11.998 | 18.052 |
| 4 | 343832 | 14.406 | 11.903 | 18.707 |
| 5 | 611952 | 10.989 | 9.639 | 19.474 |
| 6 | 328864 | 17.127 | 12.196 | 17.794 |
| 7 | 663304 | 11.794 | 8.746 | 18.999 |
| 8 | 356968 | 13.597 | 10.825 | 18.485 |
| 9 | 346040 | 11.534 | 8.768 | 17.898 |
| 10 | 425544 | 13.116 | 9.617 | 17.659 |
| 11 | 292824 | 11.511 | 9.398 | 20.8 |

Table 2. Kyber768 Metric Analysis

Here's a breakdown of what the columns in the table represent:

- Memory (BYTES): The size of the memory used for Kyber encryption (in bytes).
- Encryption Time (ms): The time it takes to encrypt a message using Kyber (in milliseconds).
- Decryption Time (ms): The time it takes to decrypt an encrypted message using Kyber (in milliseconds).
- Keygen Time (ms): The time it takes to generate a Kyber key pair (in milliseconds).

## 3.1 Results



Graph 1. Graphical representation of the metrics

**Based on this data,**

➔ For RSA we can see that the encryption and decryption times generally increase as the memory size increases. However, the key generation time appears to have a more significant increase as the memory size increases. This data provides a general idea of the relationship between memory size and encryption/decryption times for RSA.

➔ For McEliece we can see that the encryption and decryption times are generally faster than the times for RSA encryption. However, the key generation time and key size for McEliece is significantly longer than the key generation time for RSA.

➔ For Kyber we can see that the memory required is less , encryption time and decryption time is lesser than both RSA and McEliece. This is likely due to the fact that Kyber is designed to be more efficient than classical cryptographic algorithms , since it was developed with the goal of being implemented on resource-constrained devices such as IoT devices.

➔ However, it's worth noting that the key generation time for Kyber is generally higher than for RSA and McEliece. This is because Kyber uses a key encapsulation mechanism (KEM) rather than a public-key encryption mechanism, which requires more computation during key generation.

## 4. Conclusion

From the above readings and statements, we can conclude that RSA seems to be faster than the other two algorithms in terms of encryption and decryption time, but the key generation time for RSA is much larger than that of the other two algorithms. In contrast, McEliece and Kyber have much faster key generation times, but their encryption and decryption times are generally slower than RSA for small input sizes.

Kyber is a post-quantum cryptographic algorithm, which means it is designed to be secure against attacks from quantum computers. The security of McEliece encryption is based on the difficulty of decoding random linear codes, which is a well-studied problem in coding theory. While McEliece is not vulnerable to attacks by quantum computers, it is not considered a post-quantum algorithm because its security is not based on the same principles as other post-quantum algorithms. RSA, on the other hand, is vulnerable to attacks from quantum computers, which is why there is an interest in developing post-quantum cryptography algorithms.

Overall, the performance of Kyber seems quite competitive compared to the classical cryptographic algorithms, and its post-quantum security makes it a promising choice for future cryptographic applications.

## References

[1] J. Bos et al., "CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM," 2018 IEEE European Symposium on Security and Privacy (EuroS&P), London, UK, 2018, pp. 353-367, doi:
10.1109/EuroSP.2018.00032.

[2] Richter, Maximilian, Magdalena Bertram, Jasper Seidensticker, and Alexander Tschache. 2022. "A Mathematical Perspective on Post-Quantum Cryptography" Mathematics 10, no. 15:2579.
https://doi.org/10.3390/math10152579

[3] Bonnetain, X., Naya-Plasencia, M., & Schrottenloher, A. (2019). Quantum Security Analysis of AES. IACR Transactions on Symmetric Cryptology, 2019(2), 55–93. https://doi.org/10.13154/tosc.v2019.i2.55-93

[4] Bambang Harjito, Henny Nurcahyaning Tyas, Esti Suryani and Dewi Wisnu Wardani,
"Comparative Analysis of RSA and NTRU Algorithms and Implementation in the Cloud" International Journal of Advanced Computer Science and Applications(IJACSA), 13(3), 2022. http://dx.doi.org/10.14569/IJACSA.2022.0130321

[5] P. K. Pradhan, S. Rakshit and S. Datta, "Lattice Based Cryptography : Its Applications, Areas of Interest & Future Scope," 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2019, pp. 988-993, doi: 10.1109/ICCMC.2019.8819706.

[6] K. K. Soni and A. Rasool, "Cryptographic Attack Possibilities over RSA Algorithm through
Classical and Quantum Computation," 2018 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2018, pp. 11-15, doi: 10.1109/ICSSIT.2018.8748675.

[7] V. Bhatia and K. R. Ramkumar, "An Efficient Quantum Computing technique for cracking
RSA using Shor's Algorithm," 2020 IEEE 5th International Conference on Computing Communication and Automation (ICCCA), Greater Noida, India, 2020, pp. 89-94, doi: 10.1109/ICCCA49541.2020.9250806.

[8] Hülsing, A., Rijneveld, J., Schanck, J.M., & Schwabe, P. (2017). High-speed key encapsulation from NTRU. IACR Cryptol. ePrint Arch., 2017, 667.

[9] Barthe, Gilles & Fan, Xiong & Gancher, Joshua & Gregoire, Benjamin & Jacomme, Charlie & Shi, Elaine. (2018). Symbolic Proofs for Lattice-Based Cryptography. 538-555. 10.1145/3243734.3243825.

[10] Yuan, Ye & Xiao, Junting & Fukushima, Kazuhide & Kiyomoto, Shinsaku & Takagi, Tsuyoshi. (2018). Portable Implementation of Post Quantum Encryption Schemes and Key Exchange Protocols on JavaScript-Enabled Platforms. Security and Communication Networks. 2018. 1-14. 10.1155/2018/9846168.

[11] Roth, J., Karatsiolis, E., Krämer, J. (2021). Classic McEliece Implementation with Low Memory Footprint. In: Liardet, PY., Mentens, N. (eds) Smart Card Research and Advanced Applications. CARDIS 2020. Lecture Notes in Computer Science(), vol 12609. Springer, Cham.

[12] J. -C. Faugère, V. Gauthier-Umaña, A. Otmani, L. Perret and J. -P. Tillich, "A Distinguisher for High-Rate McEliece Cryptosystems," in IEEE Transactions on Information Theory, vol. 59, no. 10, pp. 6830-6844, Oct. 2013, doi: 10.1109/TIT.2013.2272036.

[13] Bucerzan, D., Dragoi, V., Kalachi, H.T. (2017). Evolution of the McEliece Public Key Encryption Scheme. In: Farshim, P., Simion, E. (eds) Innovative Security Solutions for Information Technology and Communications. SecITC 2017. Lecture Notes in Computer Science(), vol 10543. Springer, Cham. https://doi.org/10.1007/978-3-319-69284-5_10