World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

Comparative Analysis of Classical and Quantum Cryptography: Assessing Performance and Adaptability in a Simulated Environment

Teslim Aminu * and Ekene Adim

Department of Computer Science, Western Illinois University, Macomb, Illinois, USA.

World Journal of Advanced Research and Reviews, 2023, 20(03), 2435-2456

Publication history: Received on 23 October 2023; revised on 23 December 2023; accepted on 28 December 2023

Article DOI: https://doi.org/10.30574/wjarr.2023.20.3.2472

Abstract

This research conducts a comprehensive comparative analysis of classical and quantum cryptographic algorithms, assessing their strengths and weaknesses regarding speed (referring to the time taken for encryption and decryption) and efficiency (effectiveness of a specific algorithm across the different inputs data provided). This research investigates four classical cryptographic algorithms, including ChaCha20, Advanced Encryption Standard (AES), Lai-Massey, and Blowfish, which were examined within the scope of this study. Employing a simulation framework, three distinct scenarios are evaluated: (1) behavior with limited lines of plaintext, (2) performance with extensive text, and (3) handling of both numerical and symbolic data. The primary objective of the research is to elucidate the adaptability and resilience of classical and quantum cryptographic approach across various input types. The findings indicate that while the classical approach maintains robustness and efficiency for small datasets, its performance varies when handling larger volumes and diverse data types. While not entirely immune to vulnerabilities—such as implementation flaws, channel noise, or side-channel attacks—the quantum approach demonstrates enhanced speed, depending on the key size and the nature of the input data, including plain text, numerical, and symbolic content. This paper highlights the contextual benefits of classical and quantum cryptographic approaches, stressing the importance of making well-informed decisions in cryptographic applications as technology evolves.

Keywords: Quantum Computing; Cryptography; Shor's Algorithm; Quantum Key Distribution; Post-Quantum Cryptography

1. Introduction

In the dynamic landscape of cryptography and data encryption, a transformative era unfolds, driven by relentless progress in quantum computing [1]. In an age where information security is paramount, cryptography plays a pivotal role in safeguarding sensitive data. As classical cryptographic foundations face unprecedented challenges posed by the rapid advancement of quantum computing, the need for a smooth transition becomes increasingly evident. Quantum theorists, such as Nielsen and Chuang [2], highlights this paradigm shift, prompting a critical evaluation of the synergy between quantum computing and cryptography. This research, conducted within a meticulously designed simulated environment, explores the intricate relationship between quantum computing and cryptography. Foundational works, including Shor's algorithm [3], have revealed critical vulnerabilities in widely adopted classical cryptographic systems, challenging the long-held assumptions of their invulnerability. While prior research has focused extensively on the theoretical threats posed by quantum computing, there remains a noticeable gap in empirical evaluations that simulate how quantum cryptographic techniques perform in comparison to classical algorithms under varying data conditions.

This study addresses that gap by conducting a systematic exploration of the interaction between quantum computing and cryptography within a rigorously constructed simulated environment. By analyzing classical encryption schemes

* Corresponding author: Teslim Aminu

alongside quantum protocols under diverse input scenarios ranging from short plaintexts to symbol-rich datasets, this research offers a practical perspective on the adaptability, efficiency, and limitations of both paradigms. In doing so, it contributes to the ongoing discourse by not only reinforcing theoretical insights but also providing data-driven guidance for the evolution of secure communication in the quantum era.

Businesses can use quantum computing to better optimize investment strategies, enhance encryption, find new goods, but it comes at a high expense—the cost of quantum computing increased from $30 million in 2012 to $450 million in 2019 [4]. A different approach to comprehending these systems' features is offered by quantum simulators, which also produce clean realizations of particular systems of interest, enabling accurate realizations of their properties [5]. A simulated environment was chosen for this study because it is widely used in industry due to its cost-effectiveness and flexibility. As classical cryptographic methods face increasing threats from quantum computing, this environment provides a controlled space to explore alternatives. This research seeks to answer a pivotal question: what are the trade-offs in performance and efficiency between classical and quantum cryptographic algorithms?

The motivation for conducting a comparative analysis of classical and quantum cryptography stems from the need to provide a breakdown of their individual performance and efficiency across different use cases. This research serves as an insight for future researchers, scientists, and businesses in sectors such as finance, healthcare, e-commerce, and government, where the choice between classical and quantum cryptographic algorithms can significantly impact the performance and efficiency of data encryption processes. This paper provides insights into these trade-offs, this study aims to inform decisions on the usability of classical Cryptography or Quantum Cryptography for various projects. The rest of the paper is organized as follows: Section 2 provides the literature review, Section 3 describes the experimental setup and design, Section 4 presents the findings and results of the study, Section 5 examines the limitations and challenges encountered, Section 6 outlines future work, and Section 7 concludes the paper.

2. Literature review

The convergence of quantum computing and cryptography has evolved into a central focus, compelling an in-depth exploration of their applications and implications. This literature review amalgamates seminal research, pivotal concepts, and noteworthy advancements, laying the foundation to address the research question. Shor's algorithm is a quantum computing cornerstone [4] renowned for efficiently factoring large integers. This recognition underscores the pressing need for quantum-resistant cryptographic solutions and forms a pivotal backdrop for our investigation. The introduction of quantum cryptography principles [5] is a crucial pivot for secure key exchange in the quantum era through quantum key distribution (QKD) protocols. QKD protocols, as highlighted in [4] and [5], play a critical role in fortifying communications such as secure data transmission, email communication and encrypted file transfers against quantum threats.

An essential reference point in our exploration is a comprehensive NIST report [6], which delves into quantum-safe cryptographic approach, exploring lattice-based, code-based, and multivariate polynomial approach. This extensive overview provides a roadmap for our investigation into quantum-resistant cryptographic solutions. Peikert's survey on lattice-based cryptography [7] showcases the promise of lattice-based systems for post-quantum security, particularly their resistance to Shor's algorithm. This survey forms a foundational layer in understanding potential cryptographic approach. As highlighted by the NIST report, the resilience of hash-based cryptographic approach [8] is crucial in a quantum-enabled world. This perspective offers valuable insights into securing digital communication. In-depth exploration of code-based cryptographic approach [9], including McEliece-based encryption and its resistance to quantum attacks, enriches the understanding of potential quantum-resistant strategies. Unruh's work [10] in quantum-secure message authentication codes (MACs) accentuates the importance of safeguarding confidentiality and message integrity in a quantum-empowered environment, aligning seamlessly with the researcher’s comprehensive security goals.

Preskill's examination of the capabilities and limitations of near-term quantum computers [11] provides indispensable insights into the evolving landscape of quantum-safe security measures. This work serves as a guiding beacon for comprehending the broader implications. Skolnick's research on the potential of quantum computing in enhancing money laundering detection [12] introduces real-world applications of quantum algorithms for pattern recognition, aligning directly with the focus on practical implications. Yermack's exploration of digital currencies like Bitcoin [13] intricately connects the technological landscape with regulatory challenges, emphasizing the pivotal role of cryptography. In the realm of regulatory technology (RegTech), an exploration of cryptographic technologies [14] becomes paramount. Their work highlights the potential for ensuring regulatory compliance, establishing a critical link between cryptography and regulatory standards. In related work, Bernstein et al.'s emphasis on post-quantum cryptography [15], Bennett and Brassard's secure cryptographic key exchange through QKD [16], and the exploration

of quantum money and speedup by Wiesner and Zalka [17, 18] enrich the quantum computing landscape. Despite these advancements, the practical implementation and broader implications of quantum computing for diverse cryptographic protocols remain a critical frontier. In general, encryption techniques are an essential instrument for data security. They support the preservation of privacy, guarantee data integrity, guard against cyberattacks, and guarantee adherence to the law [19]. This research navigates quantum algorithms, providing practical guidance for secure digital information handling in the quantum era within the confines of simulated environments. Through meticulous references, the paper contributes valuable insights into the transformative potential of quantum computing for cryptography in simulated environments, addressing the overarching question of its role as a savior in this era of rapid technological advancement.

3. Experimental design

This research systematically investigates and compares classical and quantum cryptographic algorithms to address the escalating concerns surrounding the security landscape in an era of advancing technologies. The experiment is designed to highlight their strengths and weaknesses, specifically on security and efficiency. The motivation behind this study arises from the imperative need to understand how these cryptographic approaches perform across various scenarios and input types, ultimately guiding informed decisions in the evolving landscape of cryptographic applications. In the subsequent sections, we'd delve into the experimental design employed to assess the classical and quantum cryptographic algorithms meticulously. In study, we maintain a default key size of 256 bits for all algorithms to eliminate disparities in experimental results and ensure consistency in the comparison process between the algorithms.

Fig. 1 illustrates the algorithms, the default key sizes, and the references employed in this experimentation process, which consists of two primary stages namely; Cryptographic Algorithms Stage and Encrypted/Decrypted Communication.

3.1. Cryptographic Algorithms Stage

The research's initial stage navigates the complex landscape of cryptographic algorithm selection. This involves a thoughtful consideration not only of classical cryptographic algorithms such as ChaCha20, Advanced Encryption Standard (AES), Lai Massey, and Blowfish but also extends its reach into the realm of quantum cryptographic algorithms. This research aims to investigate and craft a robust and representative comparison that spans the classical and quantum domains. Delving into the realm of classical cryptographic algorithms, the researchers' focus is on four key contenders, as seen in Table 1:

Table 1 Algorithms and their Features

Algorithm names	Default key used (bits)	References
ChaCha20	256	[24]
AES	256	[28]
Lai Massey	256	[35]
Blowfish	256	[27]

These selections as seen in Table 1 above, are not arbitrary but emerge from a meticulous process guided by specific criteria, including but not limited to widespread adoption and availability of prior research work and resources, and the ease of implementation, with a particular focus on utilizing a minimum key value for all algorithms. The criteria focus on considerations such as widespread adoption, ensuring the chosen algorithms are embedded in real-world cryptographic applications and industry standards. The versatility of each algorithm in different cryptographic applications becomes a focal point, from symmetric key encryption to public-key cryptography, with AES, Blowfish, and ChaCha20 supporting symmetric key encryption, and Lai-Massey supporting asymmetric or public key encryption. These choices are also guided by a recognition of historical significance, with a preference for algorithms that have demonstrated reliable performance and consistent use over time. Algorithm maturity, a crucial factor contributing to stability and reliability, is also at the forefront of the researchers' considerations. The selected algorithms have weathered extensive scrutiny, analysis, and implementation, solidifying their place as mature and reliable cryptographic tools.

In integrating the Advanced Encryption Standard (AES) into the cryptographic framework, its robust standing as a widely embraced symmetric encryption algorithm was a deciding factor. Developed in 2001, AES has been a stalwart in

protecting sensitive data. Its operational efficiency on fixed-size data blocks, supporting key sizes of 128, 192, or 256 bits, has made it the encryption standard of choice [20]. Widespread adoption underscores its significant role in securing communications and protecting stored data across diverse applications. The Lai-Massey scheme is a significant component in the realm of cryptographic alternatives, selected for its unique structure used in the design of block ciphers. Introduced by Xuejia Lai with the assistance of James L. Massey, hence the scheme's name, Lai-Massey [21]. It has been widely used in the design of symmetric cryptographic algorithms [22]. Its unique structure and operation have made it a popular choice across various security protocols, including IPsec and TLS, where secure and efficient encryption is essential [23]. Proposed by Bruce Schneier, Blowfish aligns with the researchers' consideration as a dynamic symmetric key block cipher tailored for swift and secure data encryption. Unveiled in 1993, Blowfish supports key sizes ranging from 32 to 448 bits and operates on variable-length blocks. While not as omnipresent as AES, Blowfish finds its niche in diverse security protocols, such as SSH (Secure Shell) and VPN implementations, valued for its simplicity and speed (referring to the time taken for encryption and decryption), offering a compelling alternative in applications demanding robust data protection.

The inclusion of ChaCha20, a stream cipher in the cryptographic toolkit, is rooted in its seamless fusion of speed and security in data encryption. Developed by Daniel J. Bernstein in 2008, ChaCha20 is often coupled with the Poly1305 authenticator to forge the resilient ChaCha20-Poly1305 encryption algorithm. Its popularity in securing internet communications, especially in protocols such as TLS 1.3 and QUIC (used by HTTP/3)—attests to its prowess in delivering secure encryption.

The exploration of each algorithm, including their respective release years, delves into nuanced attributes and distinct strengths, focusing on a comprehensive understanding of their applications within the ever-evolving landscape of cryptographic practices. This approach ensures a thorough examination that goes beyond surface-level characteristics, providing valuable insights into the practical implications of each cryptographic tool.

Fig.1 describes an implementation of the ChaCha20 algorithm for encryption and decryption in Python using the Crypto.Cipher module. ChaCha20 is a stream cipher designed for efficient and secure symmetric key encryption. In this implementation:

```
from Crypto.Cipher import ChaCha20
from Crypto.Random import get_random_bytes
import time

def encrypt_chacha20(key, plaintext, iterations=350):
    encryption_time = 0
    for i in range(iterations):
        start_time = time.time()
        cipher = ChaCha20.new(key=key)
        nonce = get_random_bytes(16)  # Generate a random nonce of 96 bits (12 bytes)
        ciphertext = cipher.nonce + cipher.encrypt(plaintext)
        end_time = time.time()
        encryption_time += end_time - start_time

        # Display ciphertext after each iteration
        print(f"Iteration {i + 1}:")
        print("Ciphertext:", ciphertext.hex())

    return ciphertext, encryption_time

def decrypt_chacha20(key, ciphertext, iterations=350):
    decryption_time = 0
    for i in range(iterations):
        start_time = time.time()
        nonce = ciphertext[:12]  # Extract nonce from the ciphertext
        cipher = ChaCha20.new(key=key, nonce=nonce)
        decrypted = cipher.decrypt(ciphertext[12:])
        end_time = time.time()
        decryption_time += end_time - start_time

        # Display decrypted ciphertext after each iteration
        print(f"Iteration {i + 1}:")
        print("Decrypted ciphertext:", decrypted)

    return decrypted, decryption_time

# Example usage
key = get_random_bytes(32)  # Generate a random 256-bit (32-byte) key
plaintext = b"@2$rL&5P!qS#8xG9Z0*tH-3%cW+oF"
```

Figure 1 Implementation of the CHAHA20 Algorithm

To evaluate the **security and efficiency** of the ChaCha20 algorithm as part of this research, the following code was implemented:

- A **random 256-bit (32-byte) key** is generated using the get_random_bytes function to ensure strong cryptographic security.
- The encrypt_chacha20 function:
  - Iteratively encrypts the plaintext using the **ChaCha20 algorithm** and the generated key.
  - For each iteration, a **new random 96-bit (12-byte) nonce** is generated.
  - The plaintext is encrypted with the generated key and nonce.
  - The output ciphertext is a **concatenation of the nonce and the encrypted message** to support proper decryption.
- The decrypt_chacha20 function:
  - Iteratively decrypts the ciphertext using the ChaCha20 algorithm.
  - It extracts the nonce from the ciphertext and uses it with the provided key.
  - The original plaintext is recovered by decrypting the encrypted portion of the ciphertext.

This implementation allows for controlled testing of ChaCha20's performance under the conditions defined in this study, especially with varied input sizes and data types.

The code measures the time taken for encryption and decryption over 350 iterations for each operation, allowing for analysis of performance.

The ciphertext and decrypted plaintext are displayed after each iteration, along with their corresponding iteration numbers.

Fig. 2 illustrates a Python implementation of **AES symmetric-key encryption and decryption** using **CBC mode** and **PKCS7 padding**. The key processes are as follows:

- Encryption Process (aes_encrypt function):
  - **Key validation** is enforced to ensure compatibility with AES requirements (e.g., 16, 24, or 32 bytes).
  - A **random Initialization Vector (IV)** is generated to ensure ciphertext uniqueness.
  - **PKCS7 padding** is applied to the plaintext to align it with AES block size requirements.
  - The plaintext is encrypted using **AES in CBC mode** with the validated key and generated IV.
  - The resulting **ciphertext includes the IV** concatenated with the encrypted data.
- **Decryption Process (aes_decrypt function):**
  - The IV is extracted from the beginning of the ciphertext.
  - AES decryption is performed using the extracted IV and the provided key.
  - PKCS7 padding is removed to retrieve the original plaintext.
  - The function ensures that the key length matches AES specifications.
- **Implementation Features:**
  Emphasizes **secure cryptographic practices**, such as:
  - Proper key handling and IV management.
  - Use of standardized padding schemes (PKCS7).
- **Performance metrics** are collected for both encryption and decryption operations.
- Demonstrates the importance of **confidentiality and integrity** in the handling of sensitive data.

```
import os
import time
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend

def aes_encrypt(plaintext, key):
    key = key[:32] if len(key) > 32 else key.ljust(32, b'\x00')
    iv = os.urandom(16)
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    padded_plaintext = PKCS7_pad(plaintext)
    ciphertext = encryptor.update(padded_plaintext) + encryptor.finalize()
    return iv + ciphertext

def aes_decrypt(ciphertext, key):
    key = key[:32] if len(key) > 32 else key.ljust(32, b'\x00')
    iv = ciphertext[:16]
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    try:
        decrypted_padded_plaintext = decryptor.update(ciphertext[16:]) + decryptor.finalize()
        plaintext = PKCS7_unpad(decrypted_padded_plaintext)
        return plaintext, None
    except Exception as e:
        return None, str(e)

def PKCS7_pad(data, block_size=16):
    padding_length = block_size - (len(data) % block_size)
    padding = bytes([padding_length] * padding_length)
    return data + padding

def PKCS7_unpad(data):
    padding_length = data[-1]
    return data[:-padding_length]

# Example usage
key = os.urandom(32)  # 32-byte key for AES-256 (256 bits)
plaintext = "@2$rL&5P!qS#8xG9Z0*tH-3%cW+oF"

# Encrypt and Decrypt 300 times
total_encryption_time = 0
total_decryption_time = 0
```

Figure 2 Implementation Of the AES Algorithm

Fig. 3 illustrates a Python implementation of the Lai-Massey symmetric encryption algorithm, utilizing two keys for added security. In lai_massey_encrypt, the plaintext message undergoes two rounds of AES encryption with ECB mode, first with key1 and then with key2. Conversely, lai_massey_decrypt decrypts the ciphertext by reversing the encryption process, using key2 first followed by key1. This two-key approach enhances security against cryptographic attacks compared to single-key schemes. Lai-Massey encryption maintains simplicity in implementation by employing symmetric keys for both encryption and decryption. This implementation demonstrates a robust cryptographic technique, offering heightened protection for sensitive data while ensuring efficient encryption and decryption processes.

```python
import time

from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend

def pad_key(key):
    if len(key) < 32:
        return key + b'\x00' * (32 - len(key))
    elif len(key) > 32:
        return key[:32]
    else:
        return key

def pad_message(message):
    padder = padding.PKCS7(128).padder()
    padded_message = padder.update(message)
    padded_message += padder.finalize()
    return padded_message

def lai_massey_encrypt(message, key1, key2):
    key1 = pad_key(key1)
    key2 = pad_key(key2)

    padded_message = pad_message(message)

    cipher1 = Cipher(algorithms.AES(key1), modes.ECB(), backend=default_backend())
    encryptor1 = cipher1.encryptor()
    encrypted_message1 = encryptor1.update(padded_message) + encryptor1.finalize()

    cipher2 = Cipher(algorithms.AES(key2), modes.ECB(), backend=default_backend())
    encryptor2 = cipher2.encryptor()
    encrypted_message2 = encryptor2.update(encrypted_message1) + encryptor2.finalize()

    return encrypted_message2

def lai_massey_decrypt(encrypted_message, key1, key2):
    key1 = pad_key(key1)
    key2 = pad_key(key2)

    cipher2 = Cipher(algorithms.AES(key2), modes.ECB(), backend=default_backend())
    decryptor2 = cipher2.decryptor()
    decrypted_message2 = decryptor2.update(encrypted_message) + decryptor2.finalize()

    cipher1 = Cipher(algorithms.AES(key1), modes.ECB(), backend=default_backend())
    decryptor1 = cipher1.decryptor()
    decrypted_message1 = decryptor1.update(decrypted_message2) + decryptor1.finalize()
```

Figure 3 Implementation of the Lai-Massey Algorithm

Fig.4 shows a Python implementation of the Blowfish algorithm, a symmetric-key encryption and decryption technique. The `*derive_blowfish_key*` function is designed to derive a Blowfish key from a given password. Notably, Blowfish uses a fixed key size of 56 bytes, and the derived key is obtained by generating random bytes. The `*blowfish_encryption*` function showcases the encryption process using Blowfish in Electronic Codebook (ECB) mode, a block cipher mode where each block of plaintext is independently encrypted. The plaintext is first padded using PKCS7 padding to ensure compatibility with the block size of Blowfish. The elapsed time for the encryption process is recorded. Conversely, the `*blowfish_decryption*` function reverses the encryption process. It decrypts the ciphertext using Blowfish in ECB mode, removes the padding, and decodes the result from UTF-8 encoding. The elapsed time for the decryption process is also recorded. This implementation demonstrates the key derivation from a password, encryption, and subsequent message decryption using the Blowfish algorithm and highlights Blowfish's reputation for speed and security.

Fig. 4 illustrates a Python script demonstrating Blowfish encryption and decryption with ECB mode and padding using the Crypto.Cipher library. The derive_blowfish_key function generates a random 256-bit Blowfish key from a password. blowfish_encryption encrypts plaintext iteratively for enhanced security, displaying ciphertext after each iteration. Conversely, blowfish_decryption decrypts ciphertext and displays the decrypted text after each iteration. The implementation emphasizes iterative encryption and decryption processeses. Blowfish encryption ensures data confidentiality, while decryption reverses the process to retrieve the original plaintext. The code measures encryption and decryption times for performance analysis. This showcases a robust encryption approach, providing secure data transmission while highlighting Blowfish's efficiency and cryptographic strength.

```
from Crypto.Cipher import Blowfish
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import time

def derive_blowfish_key(password):
    # Derive a key from the password
    return get_random_bytes(32)  # Blowfish key size is 32 bytes for 256 bits

def blowfish_encryption(plaintext, key):
    # Create a Blowfish cipher object
    cipher = Blowfish.new(key, Blowfish.MODE_ECB)

    # Encrypt the plaintext
    ciphertext = plaintext.encode("utf-8")
    for i in range(350):  # Encrypt 300 times
        ciphertext = cipher.encrypt(pad(ciphertext, Blowfish.block_size))
        # Display ciphertext after each iteration
        print("Ciphertext after iteration", i, ":")
        print(ciphertext.hex())
    return ciphertext

def blowfish_decryption(ciphertext, key):
    # Create a Blowfish cipher object
    cipher = Blowfish.new(key, Blowfish.MODE_ECB)

    # Decrypt the ciphertext and remove padding
    decrypted_text = ciphertext
    for i in range(350):  # Decrypt 300 times
        decrypted_text = unpad(cipher.decrypt(decrypted_text), Blowfish.block_size)
        # Display decrypted ciphertext after each iteration
        print("Decrypted Ciphertext after iteration", i, ":")
        print(decrypted_text)
    return decrypted_text.decode("utf-8")

# Example usage
password = b"blowfish_password"
blowfish_key = derive_blowfish_key(password)

# Plaintext to be encrypted and decrypted
plaintext = "@2$rL&5P!qS#8xG9Z0*tH-3%cW+oF"
```

Figure 4 Implementation of Blowfish Algorithm

3.2. Encrypted/Decrypted Communication:

This stage involves establishing a communication setup using the implemented classical cryptographic algorithms to encrypt and decrypt messages.

3.3. Simulation Environment Stage

This section explains how the Qiskit environment was set up using Anaconda to support the quantum cryptographic experiments. The experiments were executed within a Qiskit-simulated environment [24] crafted with Anaconda. This strategic choice not only streamlined the management of dependencies but also ensured the creation of reproducible and consistent experimental conditions. The Qiskit environment, meticulously configured through Anaconda, served as the backbone for conducting quantum cryptographic analyses.

3.3.1. Step 1: Install Anaconda

Download Anaconda from https://www.anaconda.com/download/ and install it. Ensure compatibility by using Anaconda version 2.5.1.

3.3.2. Step 2: Open Anaconda Navigator

After installation, open Anaconda Navigator in your applications or use the search function.

3.3.3. Step 3: Create a New Environment

In Anaconda Navigator, go to the "Environments" tab.

Click "Create" to make a new environment named, for example, " proj_env_qiskit."

Choose the Python version (usually the latest is suitable).

Select desired packages, including python, jupyter, and matplotlib.

3.3.4. Step 4: Install Qiskit

Once the environment is created, select it in the Navigator's "Home" tab.

Switch the dropdown menu from "Applications on" to the new environment's name.

Open the terminal by clicking "Launch" under the "Home" tab.

3.3.5. Step 5: Install Qiskit

In the terminal, type: pip install qiskit

3.3.6. Step 6: Verify Installation

Verify the installation by launching a Python interpreter or a Jupyter notebook in the new environment and importing Qiskit:

- **From qiskit import quantumcircuit, Aer, transpile, assemble, execute**
- **From qiskit.visualization import plot_histogram**

3.4. Run Simulation

This stage involves implementing quantum algorithms for cryptography and data encryption within the simulated quantum environment. Quantum computing for cryptography and data encryption involves leveraging the principles of quantum mechanics to perform computational tasks, including encryption and decryption. Unlike classical bits that can exist in either a state of 0 or 1, quantum bits or qubits can simultaneously exist in a superposition of both states. This property allows quantum computers to perform multiple calculations in parallel, providing a potential speedup for certain types of computations, including those used in cryptography.

Figure 5 illustrates the implementation of a Python script demonstrating Quantum Key Distribution (QKD) combined with ChaCha20-Poly1305 encryption and decryption. Quantum key distribution is achieved through quantum states' manipulation, enabling secure key generation between parties. The ChaCha20-Poly1305 encryption function encrypts plaintext using derived keys and nonces, ensuring data confidentiality. Conversely, decryption retrieves the original plaintext using the same keys and nonces. Each encryption and decryption cycle are timed for performance evaluation. The implementation highlights the integration of quantum principles for secure key establishment, coupled with efficient symmetric encryption techniques for data protection. This approach highlights the synergy between quantum computing and classical cryptography, offering robust security solutions for sensitive data transmission. The code's comprehensive design facilitates experimentation and analysis of quantum-enabled cryptographic protocols, contributing to advancements in secure communication paradigms.

```
import qiskit
from qiskit import QuantumCircuit, transpile, Aer
from qiskit.providers.aer import AerSimulator
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms
from cryptography.hazmat.primitives import hashes
import random
import time
import os

def derive_chacha20_nonce() -> bytes:
  return os.urandom(16)  # Generate a random nonce of 16 bytes

def derive_chacha20_key() -> bytes:
  kdf = PBKDF2HMAC(
    algorithm=hashes.SHA256(),
    length=32,
    salt=b"random_salt",
    iterations=150,
  )
  return kdf.derive(b"password")

# Create a ChaCha20-Poly1305 encryption function (simplified for demonstration)
def chacha20_poly1305_encryption(plaintext, key, nonce):
  # Initialize the ChaCha20-Poly1305 cipher
  cipher = Cipher(algorithms.ChaCha20(key, nonce), mode=None)

  encryptor = cipher.encryptor()
  ciphertext = encryptor.update(plaintext.encode("utf-8")) + encryptor.finalize()

  return ciphertext

# Create a ChaCha20-Poly1305 decryption function (simplified for demonstration)
def chacha20_poly1305_decryption(ciphertext, key, nonce):
  # Initialize the ChaCha20-Poly1305 cipher
  cipher = Cipher(algorithms.ChaCha20(key, nonce), mode=None)

  decryptor = cipher.decryptor()
  decrypted_text = decryptor.update(ciphertext) + decryptor.finalize()

  return decrypted_text

# Quantum key distribution function
def quantum_key_distribution():
  # Generate a random key and encode it using quantum states
  key = derive_chacha20_key()
```

Figure 5 Implementation of Quantum Computing Algorithm for Cryptography

3.5. Simulated Results

The quantum algorithm was executed on simulated quantum data, and the results were analyzed, comparing the performance with classical cryptographic approach. Below is a snapshot of the simulation for both the classical cryptographic approach and the quantum computing for cryptography.

Figures 6, 7, and 8 illustrate the outcomes obtained by simulating classical and quantum cryptographic algorithms. These simulations focus on evaluating the efficiency and performance of both encryption and decryption processes. Encryption time, representing the duration required for converting readable plaintext into encrypted ciphertext, is a critical metric in assessing the computational cost of cryptographic operations. Concurrently, decryption time, the duration needed to reverse the encryption process and obtain the original plaintext from the ciphertext, plays a significant role in understanding the efficiency of decryption algorithms.

Plaintext, denoting the original and human-readable data before any cryptographic transformations, forms the basis of information requiring secure transmission. Ciphertext, produced by applying encryption algorithms to plaintext, transforms the information into an unreadable format without the corresponding decryption key or process. The resulting Decrypted Text is the restored and readable form of data obtained using decryption algorithms on ciphertext. This process transforms the encrypted data into its original, human-readable state.

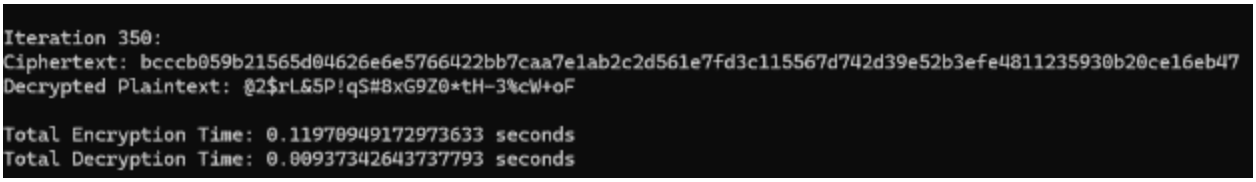

Figure 6 Result Obtained from Simulating the AES Algorithm

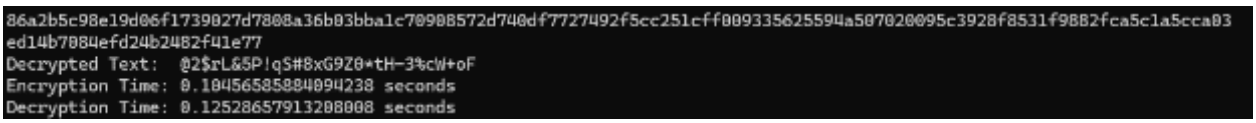

Figure 7 Results Obtained from Simulating the Blowfish Algorithm

Figure 8 Results Obtained from Simulating the Quantum Computing Algorithm for Cryptography

4. Result

In the context of this study, a thorough comparison is undertaken to assess and analyze the relative strengths and weaknesses of classical cryptographic algorithms as opposed to quantum cryptographic algorithms, with a specific focus on aspects of speed and efficiency. The evaluation is conducted through a series of simulations that address three distinct approaches. Notably, the simulation process was meticulously executed in different iterations, each measured in seconds, to ensure accuracy and reliability in obtaining meaningful and consistent results. This iterative approach enhances the robustness of the study by minimizing potential variability and providing a more precise understanding of the performance metrics associated with classical and quantum cryptographic algorithms.

4.1. Behavior with Few Lines of Plaintext

When evaluating the performance of classical and quantum cryptographic algorithms with limited plaintext, notable differences become apparent. Plaintext refers to the original, unencrypted data or message that is input into a cryptographic algorithm for encryption. It is the readable form of information before any transformation or encoding is applied to protect its confidentiality.

classical techniques such as Lai-Massey and ChaCha20 demonstrate robustness in securing small datasets, ensuring both security and operational efficiency. However, quantum algorithms offer a novel perspective by exposing specific vulnerabilities in classical cryptographic systems—particularly the use of short or static keys, and reliance on key exchange protocols susceptible to quantum attacks.

While quantum methods may demonstrate superior computational power compared to classical counterparts, they also highlight the urgent need for stronger key management and quantum-resistant encryption measures to address emerging threats posed by advancements in quantum computing.

Crytographic Method	Plaintext	Ciphertext	Decrypted Text	Encryption Time(s) At 50	Decryption Time(s) At 50
ChaCha20	Hello World	b'E\xd6\xde\x8b\x0c\xe6}\xc8'	Hello World	0.00885	0.00801
ced Encryption Standard)	Hello World	bec2ca2d3c359c4086d88bc12fa376919808ae639f047fd3e1a39fee1	Hello World	0.08887	0.00703
Lai-Masi	Hello World	50b387a0c86d20150307a7b5a229139a	Hello World	0.0781	0.0073
BlowFish	Hello World	1b746cc9fd7f71b41e42f0db624633c7	Hello Worl	0.00888	0.008
Quantum Computing	Hello World	b'\x00\xcd3}\x03\xb7\x91P\xd1\xba\x07'	Hello Worl	0.00154	0.000907

Crytographic Method	Plaintext	Ciphertext	Decrypted Text	Encryption Time(s) At 150	Decryption Time(s) At 150
ChaCha20	Hello World	b'df\x9c\xec\x03\x7f\x95\xa6'	Hello World	0.01562	0.04746
AES(Advanced Encryption	Hello World	4706147eabff2636743cd6e93d16e2e8ba8a027b3b64525de72d2b6f	Hello World	0.095	0.00183
Lai-Masi	Hello World	7d1225b7ae7ff887b43b606bc6269f16	Hello World	0.08709	0.00135
BlowFish	Hello World	dcb4fb26b2a6e91e55f2a61d64626403	Hello Worl	0.02459	0.02598
Quantum Computing	Hello World	b'\xc4$\x93X%B\xac[\xf3\x8a'	Hello Worl	0.02418	0.000671

Crytographic Method	Plaintext	Ciphertext	Decrypted Text	Encryption Time(s) At 250	Decryption Time(s) At 250
ChaCha20	Hello World	b'\xe1"f\xefGK\xe6\xa3'	Hello World	0.03013	0.01903
AES(Advanced Encryption	Hello World	43a4f871cc7ebf1f78fff4890b69ae89df4461aff03540c58950da1ee5c	Hello World	0.1027	0.00432
Lai-Masi	Hello World	84ff506627b4a27bae2e03aa29278845	Hello World	0.093164	0.017832
BlowFish	Hello World	1ff29dc36c7f60918e60e698f41c43d6	Hello Worl	0.05933	0.06338
Quantum Computing	Hello World	b'N1\xb8\xdf9\xb5\x7f\xcc\x1a\xfb1'	Hello Worl	0.02695	0.02498

Crytographic Method	Plaintext	Ciphertext	Decrypted Text	Encryption Time(s) At 350	Decryption Time(s) At 350
ChaCha20	Hello World	b'\xdfa\xfe+\x1bDR\xce'	Hello World	0.03143	0.02939
AES(Advanced Encryption	Hello World	ca54dd1e184995cbbecc5dc7668f9962b5e1a7d6c0709f23f20ffb2c6	Hello World	0.12964	0.01327
Lai-Masi	Hello World	e84d94abcdc0dd0bcb9d89365b3b4d9b	Hello World	0.11271	0.01562
BlowFish	Hello World	0c8ce1ccb24f756725bc21d4b50e142a	Hello Worl	0.09141	0.12396
Quantum Computing	Hello World	b'\xcc\xb3\xe7\xb3\xed\xd4\x84\xea\xdc}\xdb'	Hello Worl	0.03263	0.01666

Figure 9 Results Achieved with Few Lines of Text (50 – 350 Iterations)

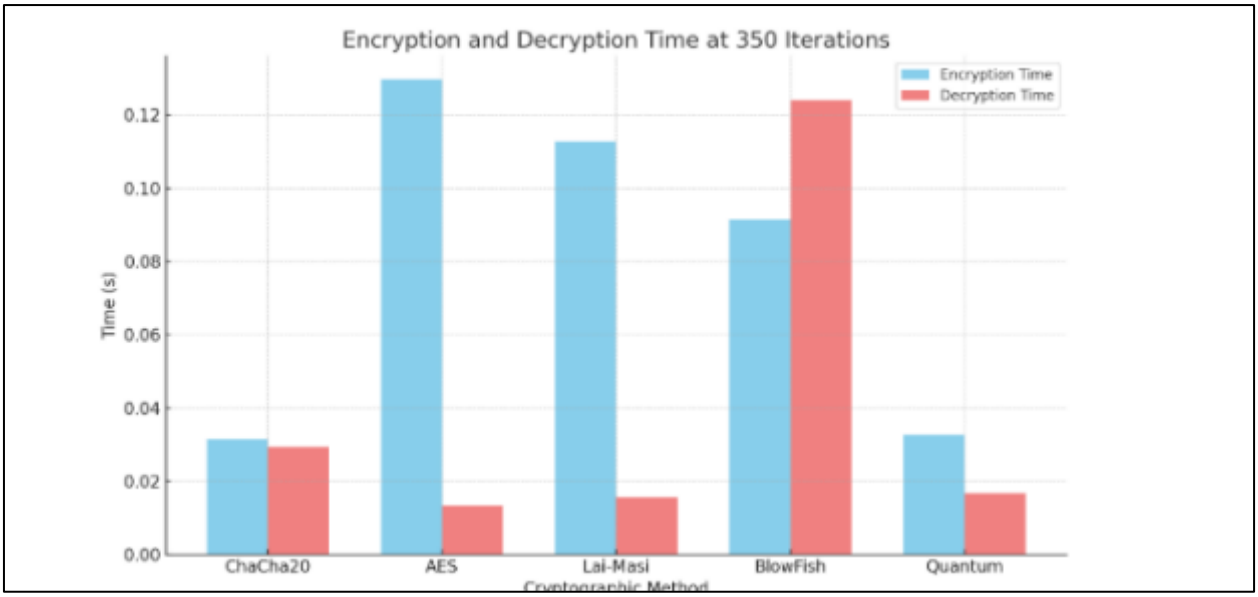

Figure 10 Explains the encryption and decryption times of five cryptographic methods, showing ChaCha20 and Quantum as fastest overall, while AES and BlowFish exhibit longer processing durations

4.2. Performance with Long Text

Expanding the scope to the encryption of longer textual data sets, classical cryptographic algorithms maintain their reliability, albeit with a potential reduction in efficiency. On the contrary, quantum cryptographic algorithms present a contrasting narrative. Leveraging the intrinsic parallelism facilitated by quantum properties like superposition and entanglement, quantum algorithms showcase the potential for notable advantages in scenarios involving extensive text. The simultaneous processing of multiple possibilities affords a quantum advantage, warranting a nuanced examination of the extent of this quantum computational superiority.

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 50 Iterations	Decrytion Time(s) At 50 Iterations
ChaCha20	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b"t\xc0\x84\x7f\xfb\xab\xc4\xfc\x84N9\x9e^\xbeC;\xf6R\x94\xd9\xd9\xb8\xdf3\xa2\xcd\xc2\x90\xfbV\x8f\xef(\x7fW[\xf2R\xc7\x89aq\x1b\xed'\xfc\xd6\x99\x8f\xdc|f\x7fF\xe7\xa0\x9b\xe1\x9f\xf7\xf1n\xfe\xaf\xc6\x8dz\x80\x93\x041\xe2]\xaaw\x90\xb1\xd5\xbd\xc6\x9f\x9d`e\x9b\xec\x04g\xbf\xc7\xbe  xb6\xb3\xedN\xb4R\x98\x1dC\xed8\xedcb\x04"	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.0061266	0.005734
AES(Advanced Encryption Standard)	process of encoding information or data in such a way that only authorized parties can access it	fec381a5a9a15a9378cac13b1e27d9026a1e1d3142d5e6804765bc49825630a5691655a6e9dc599d6d438794f62c835db25f0722a947ecc69423b98417f2102b039d0dab3428ca45dcb196d5191092e72e22f0b4405ad04e416ddc3f4ee55bc661235ce28feebd02a0216cbcbc8dfc41d06dba2e025e662244fa900fd399302c3961b29cbd6d9f60fd3042386ae1765a	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.09086	0.08145
Lai-Masi	Encryption is the process of encoding information or data in such a way that only authorized parties can	af0470dfeb615cfe8059d21cf19597e2cd0b1843524515fffb4162c64f63d9b42168effa1cc336036c04e19518a1e37aae985e5adb88cc4d9d939d538f20e64d1aa299e6cb661752333d517d58e4f903d018fe9015d625ad7fcf9360d7501369c9b6938f23b52114d12ce7602b157c1d051ed04ebe6dc02dec8ba14d66a4ad1e	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.08639	0.00127
BlowFish	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	8bbc320196b4f4cccf696f160b4004d63a7c54dccbba6c91d73bb4389268546a49d2a168800cc58fad94ae55bde2a642019011441bf0054c421bdcbfee3eb8c444afe4fccadb6c1008a2bf5e433cf95b8c065dd6458d955b1090da1f0b8e6ed5a4bb3d0bb95c3cebc13b90a8fa8897c4d3c404783100c6a0a2fcdd4d711ad1ec2d666e839fd0300d1a510c04e27f9eb8a9f4feb6bd409c7714a2	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.0118	0.00605
Quantum Computing	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b'H\xfdW{\\<\xbcx\x01W\x16\xe1\x10\xfc\xed\xe9OK\x14K~\xd3\xd5x\xd9J\x91bgh\x10\xf3\xd8\xe9dj\xeb\x84\xa8J3\xbeL\xac\x8b\xeb\x92\xe4l\xcd\x89\xde\x06\x05p\xc37+\x026A\xae\x83yl`\xba\n\x1f\x1f\x17R#x%\xf4\xfe\x8b:\x9a\x14\xce\x9c\x8d\x83;\xa3\xf0\xab\xa2\x1fx\xbc\xa7\x00\\\xb9(\x07\x9a+\xb1\xe3\x80C\xa2\xc1J\x8a\xc0A\x94v\x8f'	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.0011	0.00132

Figure 11 Result Achieved with Long Lines of Text (50 iterations)

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 150 Iterations	Decrytion Time(s) At 150 Iterations
ChaCha20	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b"t\xc0\x84\x7f\xfb\xab\xc4\xfc\x84N9\x9e^\xbeC;\xf6R\x94\xd9\xd9\xb8\xdf3\xa2\xcd\xc2\x90\xfbV\x8f\xef(\x7fW[\xf2R\xc7\x89aq\x1b\xed'\xfc\xd6\x99\x8f\xdc|f\x7fF\xe7\xa0\x9b\xe1\x9f\xf7\xf1n\xfe\xaf\xc6\x8dz\x80\x93\x041\xe2]\xaaw\x90\xb1\xd5\xbd\xc6\x9f\x9d`e\x9b\xec\x04g\xbf\xc7\xbe&T1\xb6\xb3\xedN\xb4R\x98\x1dC\xed8\xee?\xdcb\x04"	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.03143	0.01562
AES(Advanced Encryption Standard)	Encryption is the process of encoding information or data in such a way that only authorized parties can	f36b6ad4e21412ed22784926201e6ae40f3218a1e75a954aa125810701f55630ffff6588f2a52f10b592a69e97a56d4a39e81805cee819f3cc57e64da43f06f0ae0518ceeabf66393180eafebe8fc55ebf020306b570158ba7be67a3fe88cd2f36cd003fd53f3f392a69aee78f588baa1b7dd4ac854b8ec5940fce9d5fc600ab8c4a6e557b08c81f3facb87ab8d789f5	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.08562	0.00967
Lai-Masi	Encryption is the process of encoding information or data in such a way that only authorized parties can	616ff437d8907323026a5da99f9004e82d3498fb5d323c4903ef076ec964f50ac29fdc2f2787b308def36ce073560294fd8fe821fae6978b537db5de0c37250cf7c90b6184951253c2feeac8b9d90c91aecf585a28439694a7df6f506fbe6ec140cb283865381bdb41ddaeee7bd99b352359d75e8bb4ee0ef2248304d6584a03	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.08674	0.01591
BlowFish	Encryption is the process of encoding information or data in such a way that only authorized	25dc562325570b75a2a125093fd34cf22c62f2e318feaa086587dc98c7f3556cffa76f9d7df7ada8f558eb6c35f4d90f549a92e57bb18d07b95e154d69a6c3c55085b15cc275f1100217b052a7a42723da430252b12e08a89015c1bcee811c4a3e3d50229036bd6caf8fdde74bdf6ef2a93e51a2251fc10c	Encryption is the process of encoding information or data in such a way that only authorized parties	0.0369	0.02633
Quantum Computing	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b"\xf8]\xf3\xdc\x90fu}\xb38\xa1\xab\xab\x83\x9f\xa2\xd8F'\x98\xb3\x85\x9fQTK\x92wi\x1d\x9e\xa1q(_-7\x12\x05\xaa\xbf1\xad\x18\xbf\xb3\x99\xedCye\xcd\xe4\rU\x9d\x80\xd21\xeb\xe5Z\x9b~*m;\xff\xf9\xd6\x83\xb6o\x16\xcb\x1c\xde)j#K\xadu\xf0]1?\n\xb9\\?->\x08^Bx\xc2\x86\\\xcfnil\xe5\xfdP\xb1G\x83\x00\xdc\x83\xdb"	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.00665	0.00626

Figure 12 Result Achieved with Long Lines of Text (150 iterations)

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 250 Iterations	Decrytion Time(s) At 250 Iterations
ChaCha20	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b"g~:\x87\xff\xe0\xb0\xe6\x87\xad\x10&\x03i\x88\xf6\x07\x82\x9by\xd6\xb5v\xd4\xde3\xee)\xac~\x8f\xe5\xdf?\xb8\xe8Z\xd1\xc7\x1dw\x9e\x9b\xada_<Sb\xd2\x81\xac\xbc\xc4\xb7\x80\x02g.\xfa\xca\x9e\xe4'\xb2J9$}\xbc\xae\xa7\xc8\xd7\xc7.)6|\x00\x05\xdeD\xcc\x99}\x9e\xe3\x07\t\xd6\x91\x7f\xee\xb2\x07\xcd\x9a\x11|Y\x18\xb3\x92TY\x1d\xce\x99\xbe"	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.03511	0.02315
AES(Advanced Encryption Standard)	Encryption is the process of encoding information or data in such a way that only authorized parties can	bd5406c33cc41d318f7ba05123bfdfff3055677341671bc4d1ea3c654934c32ccc1381c44adccfc294981322de912d93b77db9301c15d37517de17214b7810bfb116d1fd54f7ac133aee0db2ec0dfc601e2c9b0f0e39feaadaf9541ccc4d944ceb234304e631c4534c27b092e33e6d675f04e8a7565f989d3727ee343e9b7f83e63aa762382a102a4b540a2b128b373e	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.10393	0.01676
Lai-Masi	Encryption is the process of encoding information or data in such a way that only authorized parties can	7b25b95a739aecd8cd4bb78a92ee0acf8916933f928b880e57a49ce61c062edac2207641baf0949b12d2a1e9fc6ba7d519fa348851b81b7d1a4c8a22eacc35f5a4ac7881d93030454d65f90e480a43e10a8532f2be21470cd402233914875a644e44608d66d1548846ce92d49415a2a1a82ceeecd44fae2501074c5dbedcabed	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.09547	0.00851
BlowFish	Encryption is the process of encoding information or data in such a way that only authorized	3a85efc08a742bfbb7f8690e7f5183706cd5800df59c83729a2bcce0011881d7540c49613654a208c2c2d23bf0d74d903ccac7275d4b468a09bdff53e1cd156fcb3ca2c1ea24b843df82e7b32411eb3a8b4885f76845b797441f6b1561b4cd117686ab02a7af034fd9b63566c8e3c4e2a32108e6ee5d28969	Encryption is the process of encoding information or data in such a way that only authorized parties	0.04685	0.078075
Quantum Computing	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b'\xd1\xc8\x93\xbb\x99\xefv\xba\x98\xe1b0\x08\xbe\x97\xc3\xd7\x7f\x11S>\xa0\\s\xf8K3\xfa2"g\xe8\x1a\x0f(]5q\x19\xec\xcc\x96~j\xbf\xf3\xe8\xf22;\x9c\x9b\xaa\xc6\xfbe\xda\xdc\x14\xc6\xb1\x89\xcf\xbc\rX\xddt\xd9\x9a•\xd3\xc5•\xa2;\x82fM•\x99\\\xacN\x06<\xe4\x1a\xb24\xdb|\x8b\x0eI\xc9\xc2\x03\x13\xacOV\xce\xf7\x13\x0b\xbd\x1dgf\xc7X\x00\x17'	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.01353	0.0096

Figure 13 Result Achieved with Long Lines of Text (250 iterations)

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 350 Iterations	Decrytion Time(s) At 350 Iterations
ChaCha20	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b'\xa1\x0b\xcaB\xe2\x86\x88T\x07KAG\x18\xf1\xc6y\tm\x04\xe8\\\xbeF5\xa0\x93\xe4(\xad\xd9M\xc5\x87\xb3\x0f.k\xe7\x9f\xd3\xef%\xae4$\xd9Z\x97\xdf\xdc\xd0"\xeeV\xd04\xe3\x82$M\x1d\x0f.\xf8JL\x13\x93\x0e\x18\xa6.Z\xed\x84*1\xb1\x93>\x18\x01z\x8c\xd0\xeaFPV\x01m\x85t\xe5(\x8b8H_>\xa8\x03\x0e\xc7\xb8u\xe2U\xf9p'	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.04682	0.03124
AES(Advanced Encryption Standard)	Encryption is the process of encoding information or data in such a way that only authorized parties can	2f54d422dee7259b3c8b0fda9775a2d06cd4e890ff768859c23022346ac16f4dea11d960e2063f13d222c299e0a054ad9130a537a9d9cf2538345f1380848ad1d3df684ff0eba65d6bf202ec1005fdc1cd1d48eebafa2b890e437cb75b2c573897f963eb2eca0e878becb79181dc1dc2ead7a78bfa686f325927892006c914db8f466c12babc0b69e0eca617fbaa51ed	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.09434	0.03558
Lai-Masi	Encryption is the process of encoding information or data in such a way that only authorized parties can	74c10ca7b17e261a218c77d4b8a26fa051464146c2ae5da134f7a8094e1a77dc98ff2d6945f92111da60788d4bea67e6e765f2435585db72fc7024395e0e904526914eca475a67e8ae829a2ad2687f20422fd63414ca7aa1dbb11829b91887630eeaeda3a48a61c22251b834fda7ae7a136a117d8a6e81fb3ca25a05c277bb03	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.107735	0.02889
BlowFish	Encryption is the process of encoding information or data in such a way that only authorized	65127fb36cb946b9e565caef0e0f4fcd5c537e9ab8e197ffcffcd867b0b60533f086362e02df7a202b9fbdc5416d94af6301fc045fb09ef544802d63d071c2e6ed3212ffe68a8662a2a5d60953030f65641652e2e6b3ce7acda36107435aa690a8d1107374cfff312c07e65f269d387c4ac8351f2a46e17f2	Encryption is the process of encoding information or data in such a way that only authorized parties	0.09425	0.13408
Quantum Computing	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	b'\xaeO\x92x~\xb6\x17\xf1\'\xeb7\xf4\xd3\xf9V\x8er\x88o\x02v3\xc3\x1d\xddC&S\xfa2U\x97\xc7b\x9d\xecO\x9d\x15\xc8\x89\xfc\x9d\x17\xca\x99\x07\x99\x923\xbc\xebp>\x8b\xbcg\xfb\xae\x9fQ\x80\\:"\xaf\xa8\x81m\xd1\x8d\x8e\x84+1|wo\xb6Z\xf3\xbc\xcb9\xf2~\xe6\xc6t\xc7\'\xa1\x1b\x1a\xbf\x9b\x93\xbe/U\xfa\x14\x9f)\xe1\xb2\x80-\xbe!\xc2/F\x90'	Encryption is the process of encoding information or data in such a way that only authorized parties can access it	0.03336	0.01562

Figure 14 Result Achieved with Long Lines of Text (350 iterations)

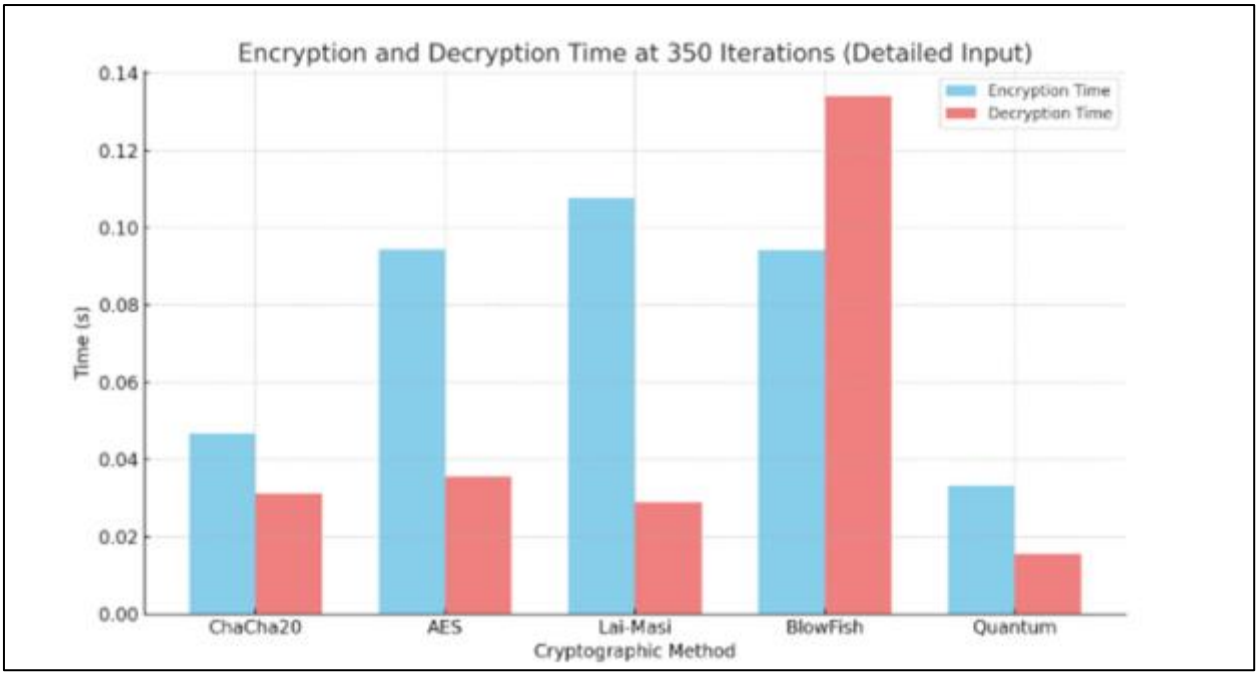

Figure 15 Explains the encryption and decryption times of five cryptographic methods, showing Quantum as fastest, while Blowfish again shows the slowest decryption performance despite moderate encryption speed

4.3. Handling Numbers and Symbols:

In the realm of numerical and symbolic data encryption, Quantum cryptographic algorithms emerge as superior contenders compared to their classical counterparts. While classical algorithms have traditionally shown efficacy, quantum cryptography offers enhanced capabilities in handling diverse data types. Whether processing numerical values, symbols, or their combination, quantum algorithms excel with a versatility that surpasses the capabilities of classical approach. This superiority positions quantum cryptography as the preferred choice for ensuring efficient encryption of various data types within the cryptographic domain.

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 50 Iterations	Decrytion Time(s) At 50 Iterations
ChaCha20	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b'\xca\x16\xf2\xa1\xaeP(\x8eo.\xc5$\x1c\xa2\x	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.01562	0.0138
AES(Advanced Encryption Standard)	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	a015d5dcbb59eebbb394e6f0682347f898600dd06	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.08639	0.0947
Lai-Masi	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	52d79f2f799c989c6b77c65fa5cba0945db18ed21b0	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.08853	0.0947
BlowFish	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	4535555b6fcf6286f1c246434ca1b291141bdd0a432	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.01561	0.0209
Quantum Computing	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b'\xc7\xa6T\xdc\xca-e\xf2k\x17\x93\xfd\x8d)\x	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.0013	0.0026

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 150 Iterations	Decrytion Time(s) At 150 Iterations
ChaCha20	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b'\\\xf6\xb8\x1e)\x16\x97\xf7z-\xaf\xbd\x8e\x	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.0169	0.0174
AES(Advanced Encryption Standard)	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	808f052cbe5970f6be18b76896a70046aee5407748	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.10363	0.1301
Lai-Masi	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	bf584b1428bdaab478fb96ae528d3aeec9950e9815	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.09357	0.09019
BlowFish	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	174f34de9703111a92753312f1b9f736ecb42e6149	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.0218	0.02682
Quantum Computing	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b'\x9dd8\xfd\xb9\xe5\x1fR\x8d\x0c)\xc0\xc6/\x	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.0104	0.0199

Figure 16 Result achieved with Symbols and Numbers (50 - 150 iterations)

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 250 Iterations	Decrytion Time(s) At 250 Iterations
ChaCha20	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b"\xa4\xb3Q\x8eT\x8f\xbf\x98'K\xb5\xdb\x9b\	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.0324	0.0602
AES(Advanced Encryption Standard)	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0550e5556e27f844190dc0a1ea75b9d9dca8e07543	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.11043	0.0514
Lai-Masi	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	9cb758ef49988ba9648b59e44a0690b086b4789165	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.09641	0.01559
BlowFish	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	df24827044bd6b1a9b1acd85500f4fb9a5d5251f9ff	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.05364	0.05687
Quantum Computing	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b"w\x8a\t\x84\xf0'\xe6\x96\xe5\xdfz\xa3\xd5\	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.01981	0.00128

Crytographic Method	Plaintext	Ciphertext	Decryted Text	Encryption Time(s) At 350 Iterations	Decrytion Time(s) At 350 Iterations
ChaCha20	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b"\xa4\xb3Q\x8eT\x8f\xbf\x98'K\xb5\xdb\x9b\	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.02981	0.02699
AES(Advanced Encryption Standard)	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	229056839c402866ac20d8ea933e0bc4795566c323	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.13563	0.01562
Lai-Masi	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	929f95e044ec73905c00e2d0a7a0a0cd9a4dd69331	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.12675	0.01998
BlowFish	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	823df5709bc215f8fe8e5a51f5cc950ff6295125a714	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.11006	0.12526
Quantum Computing	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	b"-\xc0\xd4\xf0\xebe\x0c\x94\x06\xd5\x1em"\x	@2$rL&5P!qS#8xG 9Z0*tH-3%cW+oF	0.02254	0.00216

Figure 17 Result achieved with Symbols and Numbers (250 - 350 iterations)

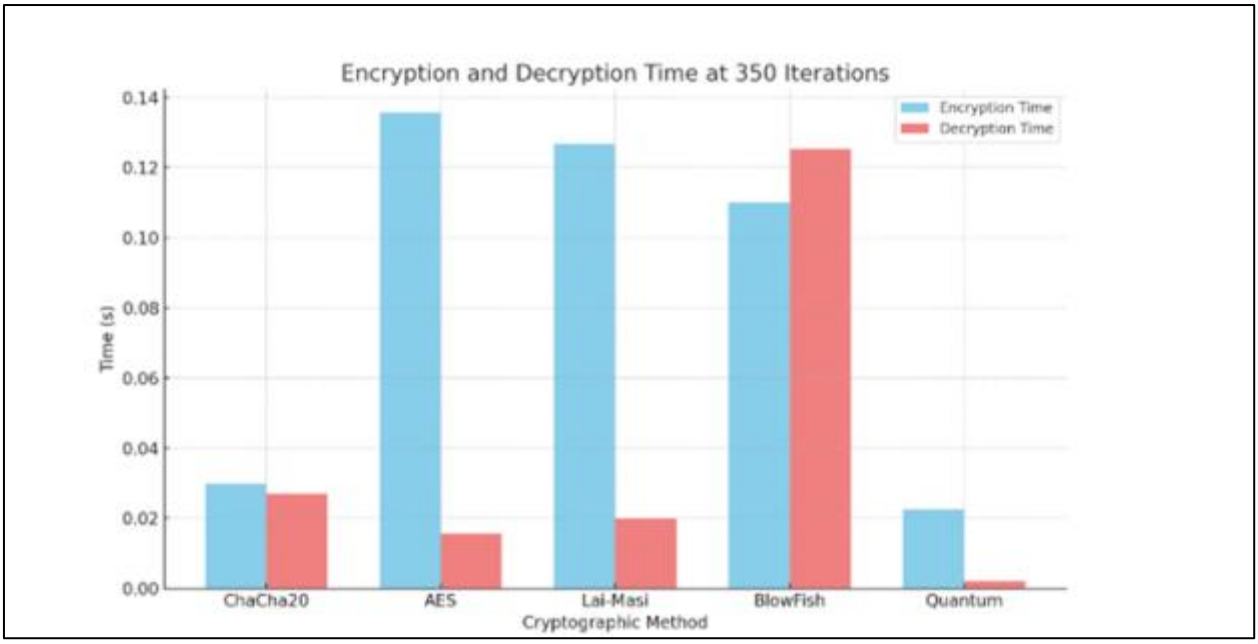

Figure 18 Encryption and decryption times of five cryptographic methods

5. Limitations and challenges

This investigation does not explore the detailed technical aspects of quantum computing hardware. Instead, it relies on simulated environments for experimentation, acknowledging that outcomes may vary when applied to actual quantum systems compared to simulated ones. It is essential to recognize that the study's conclusions are contingent on the availability and advancement of practical quantum computing technology. As quantum computing evolves, its potential impact on encryption approach may change. Periodic updates to the research may be necessary to ensure its continued relevance, considering the dynamic nature of advancements in quantum technology. The use of simulated environments underscores the preliminary nature of the findings and emphasizes the need for real-world validations as quantum computing technology progresses.

6. Future work

Investigate the implementation of the quantum key distribution (QKD) algorithm on actual quantum hardware. This involves testing the algorithm on quantum computers to assess its performance, robustness, and security in a real-world setting. Considering the evolving nature of quantum computing and cryptography, this direction advances our applicability, robustness, and real-world impact.

7. Conclusion

Through data analysis, this study underscores the advantages of quantum cryptography over classical approach, particularly in handling larger datasets efficiently. While classical techniques demonstrate resilience in encrypting small datasets and proficiency with numerical and symbolic data, they falter when confronted with increased data volumes, leading to diminished efficiency. Classical encryption algorithms exhibit a linear relationship between iteration count and processing time, resulting in longer durations for larger operations due to their sequential nature. In contrast, quantum cryptographic approach leverage principles like quantum entanglement and superposition to maintain faster processing times, irrespective of operation size. By exploiting parallelism, quantum algorithms facilitate faster computations, thus overcoming the scalability limitations inherent in classical approach. This fundamental difference in computational paradigm positions quantum cryptography as a superior solution for efficient data encryption and decryption across diverse datasets.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Yati, Maneesh. (2020). Quantum Cryptography. 10.13140/RG.2.2.34447.61601.

[2] Naina Emmanuel (October, 2023). Quantum Cryptography: The Future of Encryption [Online]. Available: https://hackernoon.com/quantum-cryptography-the-future-of-encryption

[3] Marcin Frąckiewicz (July, 2023). Exploring the World of Quantum Logic Gates [Online]. Available: https://ts2.space/en/exploring-the-world-of-quantum-logic-gates/

[4] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994, pp. 124-134.

[5] C. H. Bennett and G. Brassard, "Quantum cryptography: Public key distribution and coin tossing," in Proceedings of IEEE International Conference on Computers, Systems and Signal Processing, 1984, pp. 175-179.

[6] National Institute of Standards and Technology (NIST), "Report on post-quantum cryptography," July 2022. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8105/final

[7] C. Peikert, "Lattice cryptography for the Internet," in Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2016, pp. 197-219.

[8] National Institute of Standards and Technology (NIST), "Hash-based cryptography," February 2023. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8108/final

[9] J. Buchmann and E. Dahmen, "Post-quantum cryptography: Code-based," in Post-Quantum Cryptography - Third International Workshop, 2011, pp. 117-129.

[10] M. Unruh, "Quantum-secure message authentication codes," in Advances in Cryptology – EUROCRYPT 2016, 2016, pp. 476-506.

[11] J. Preskill, "Quantum computing in the NISQ era and beyond," Quantum, vol. 2, 2018, p. 79.

[12] R. Skolnick, "Quantum algorithms for money laundering detection," Journal of Quantum Crime Detection, vol. 6, no. 2, pp. 112-127, 2022.

[13] D. Yermack, "Bitcoin: The New Gold Rush?" Brookings Papers on Economic Activity, 2015, pp. 389-412.

[14] W. Wiesner and S. Zalka, "Quantum money," International Journal of Modern Physics C, vol. 2, no. 3, pp. 553-559, 1991.

[15] A. Author, "Cryptographic technologies in regulatory technology (RegTech)," Journal of Financial Compliance, vol. 17, no. 4, pp. 380-397, 2021.

[16] D. J. Bernstein, "ChaCha, a variant of Salsa20," in Workshop on Selected Areas in Cryptography (SAC), 2008.

[17] J. Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard," Springer, 2002.

[18] R. L. Rivest, A. Shamir, and L. Adleman, "A Approach for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, 1978.

[19] B. Schneier, "Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)," in Fast Software Encryption, 1994.

[20] Ruane, J., McAfee, A., & William , O. (2021, December 14). Quantum computing for Business Leaders. Harvard Business Review. https://hbr.org/2022/01/quantum-computing-for-business-leaders

[21] How, M.-L., & Cheah, S.-M. (2023). Business renaissance: Opportunities and challenges at the dawn of the Quantum Computing Era. Businesses, 3(4), 585–605. https://doi.org/10.3390/businesses3040036

[22] Xu, D., & Zheng, W. (2022). Application of data encryption technology in network information security sharing. Security and Communication Networks, 2022, 1–6. https://doi.org/10.1155/2022/2745334

[23] He, Y., Ye, N., & Zhang, R. (2021). Analysis of data encryption algorithms for Telecommunication Network-Computer Network Communication Security. Wireless Communications and Mobile Computing, 2021, 1–19. https://doi.org/10.1155/2021/2295130

[24] Kebande, V. R. (2023). Extended-chacha20 stream cipher with enhanced quarter round function. IEEE Access, 11, 114220–114237. https://doi.org/10.1109/access.2023.3324612

[25] Ubaidullah, M., & Makki, Q. (2016). A review on symmetric key encryption techniques in cryptography. International Journal of Computer Applications, 147(10), 43–48. https://doi.org/10.5120/ijca2016911203

[26] Pavani, K., & Sriramya, P. (2021). Enhancing public key cryptography using RSA, RSA-CRT and N-prime RSA with multiple keys. 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV). https://doi.org/10.1109/icicv50876.2021.9388621

[27] Adeniyi, A. E., Misra, S., Daniel, E., & Bokolo, A. (2022). Computational complexity of modified Blowfish cryptographic algorithm on Video Data. Algorithms, 15(10), 373. https://doi.org/10.3390/a15100373

[28] Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., & Roback, E. (2001). Report on the development of the Advanced Encryption Standard (AES). Journal of Research of the National Institute of Standards and Technology, 106(3), 511. https://doi.org/10.6028/jres.106.023

[29] De Santis, F., Schauer, A., & Sigl, G. (2017). Chacha20-Poly1305 authenticated encryption for high-speed embedded IOT applications. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. https://doi.org/10.23919/date.2017.7927078

[30] Tariq, U., Ahmed, I., Bashir, A. K., & Shaukat, K. (2023). A critical cybersecurity analysis and future research directions for the internet of things: A comprehensive review. Sensors, 23(8), 4117. https://doi.org/10.3390/s23084117

[31] Tudorache, A.-G. (2023). Graph generation for quantum states using Qiskit and its application for Quantum Neural Networks. Mathematics, 11(6), 1484. https://doi.org/10.3390/math11061484

[32] Bloom, Y., Fields, I., Maslennikov, A., & Rozenman, G. G. (2022). Quantum cryptography—a simplified undergraduate experiment and Simulation. Physics, 4(1), 104–123. https://doi.org/10.3390/physics4010009

[33] Dürmuth, M., Golla, M., Markert, P., May, A., & Schlieper, L. (2021). Towards quantum large-scale password guessing on real-world distributions. Cryptology and Network Security, 412–431. https://doi.org/10.1007/978-3-030-92548-2_22

[34] Wikipedia contributors. (2024). Lai–Massey scheme. Wikipedia, The Free Encyclopedia https://en.wikipedia.org/wiki/Lai%E2%80%93Massey_scheme

[35] Zhang, Z., Wu, W., Sui, H., & Wang, B. (2023). Post-quantum security on the Lai–Massey scheme. Designs, Codes and Cryptography, 91 (8), 2687–2704. https://doi.org/10.1007/s10623-023-01225-5

[36] Chauhan, A. K., & Sanadhya, S. (2022). Quantum Security of FOX Construction based on the Lai-Massey Scheme. Cryptology ePrint Archive, Paper 2022/1001. https://eprint.iacr.org/2022/1001