Received November 9th, 2024; accepted December 23th, 2024. Date of publication December 30th, 2024
Digital Object Identifier: https://doi.org/10/25047/jtit.v11i2.5517

An Encryption Method of 8-Qubit States Using Unitary Matrix and Permutation

RIZKY ALFANIO ATMOKO[1], ERIK YOHAN KARTIKO2, AGUNG TEGUH SETYADI3

1Universitas Jember, Jember, Jawa Timur, Indonesia
2 Universitas Jember, Jember, Jawa Timur, Indonesia
3 Politeknik Electronika Negeri Surabaya, Surabaya, Jawa Timur, Indonesia

CORESPONDING AUTHOR: RIZKY ALFANIO ATMOKO (email:rizkyaatmoko@unej.ac.id)

ABSTRACT The paper explores the methods for encrypting and decrypting an 8-qubit states of quantum system using unitary and permutation matrix. Our approach utilizes a unitary matrix to create a new superpositions of an encrypted 8-qubits states. By applying a permutation matrix, we shuffle the state vectors, adding an additional layer of security. The encryption process will be performed on the encrypted state X using the formula $X' = X \cdot U \cdot P$, where X is the original state vector, X is the unitary matrix, and P is the permutation matrix. To ensure the total probability remains normalized, we showed that the resulting new 8-qubits state X' remains normalized. The decryption process is achieved by applying the following operations, $X = X' \cdot P^T \cdot U^\dagger$ retrieving the original state. This paper also is showing that the original quantum state can be accurately recovered post-decryption. This highlights the robustness of our approach in maintaining the integrity of quantum information. Furthermore, we aim to create n block for n different 8-qubits state using a different key in each block from the initial unitary matrix U and permutation P. In order to implement these methods, we need to generate a new unitary matrix for each block. Either by random pick or using iteration. In fact, we showed how to create the new unitary matrix using iteration for each block. Here we showed that the new generated matrix UP is also a unitary matrix so that we can use iteration proses to create a new unitary matrix in each n block for n different 8-qubits state. Here we generate the unitary matrix U_n from U_{n-1} as key in block n. This result in the encryption of each block for each 8-qubits state using the formula $X'_n = X_n \cdot U_n \cdot P$ resulting in a more robust security. The encryption/decryption scheme we referenced can theoretically be implemented on modern quantum hardware but verifying operations involving hundreds of qubits would demand rigorous calibration and error correction.

KEYWORDS: Qubits, Unitary Matrix, Hadamard Matrix, Permutation Matrix, Superposition, Cyber security

I.INTRODUCTION

In 1982, R. P. Feynman [1]] introduced an interesting concept that laid the groundwork for modern quantum computing. His exploration of simulating physical processes with computers proposed that classical computers might be inadequate for simulating quantum phenomena efficiently. The surge in quantum computing has revolutionized how we approach data encoding and security such as Shor's Algorithms [2] to break a certain security system that previously was considered complex. Unlike traditional digital frameworks that rely on binary digits (0s and 1s), quantum systems leverage qubits. These qubits can exist in 0, 1, or any complex superposition of both, enabling more intricate and powerful data operations [3]. To represent alphabets (like A-Z), quantum computers use encoding methods that map each letter to a unique binary sequence, similar to ASCII encoding in classical systems. However, in quantum systems, these binary representations can be placed into superposition states, allowing for more complex operations [4]. Qubits can represent alphabets by encoding them in states that combine multiple quantum states simultaneously. The multiple qubits also explained in [4] including the 8-qubits system.

In classical computing however, letters are represented using 8-bit binary sequences. Conversely, in quantum computing, the concept shifts to an 8-qubit system. This system consists of 8 individual qubits, each capable of existing simultaneously in a superposition of the states $|0\rangle$ and $|1\rangle$. This superposition allows quantum systems to encode information in a more various way than classical bits, enhancing both the complexity and potential capabilities of data representation and manipulation within quantum system. As the number of qubits increases, the dimension of the state space increases exponentially. As explained before in [3], in general, an 8-qubit system has a state space of

$$|\Psi\rangle = \sum_{i=1}^{255} c_i|i\rangle$$

The $i\rangle$ in here represent one of the 256 possible basis state combinations of the qubits, and c_i are complex coefficients corresponding to the probability amplitudes for the basis state $i\rangle$ and each c_i must satisfy the normalization condition:

$$\sum_{i=0}^{255} |c_i|^2 = 1$$

For example, a letter could be mapped to a specific quantum state that corresponds to a binary encoding like '01000001' for 'A'. In this case, the c_i of '01000001' will tell us the probability this 8-qubits will result in 'A' when measured.

In [5] the classical cryptography and some of the coding theory and method were explained. One fundamental area of study involves the development and enhancement of classical encryption methods to fit within quantum system. This paper explores the combination of unitary matrix and permutation transformations to encode 8-qubit states in matrix form, drawing connections between traditional binary code theory and quantum system.

Binary code theory underpins much of classical data encryption, such as in [5] where data is represented and manipulated using sequences of binary digits also knows as bits. As a simple example, shift cipher, or Caesar cipher, is one of the simplest and oldest encryption techniques. It works by shifting each letter of the plaintext a fixed number of places in the alphabet.

In the realm of binary code theory, our encryption method adapts with traditional encryption methods while incorporating layered security strategies. Previous works, such as the Hill cipher [11], which also be explained in [5], is a matrix transformation cipher that uses linear algebra principles. The plaintext is divided into blocks of letters, and each block is represented as a vector. These vectors are then multiplied by an invertible matrix (key matrix) to produce ciphertext. In the binary context, this method can be adapted to handle matrices over binary fields, extending its applicability to digital data encoding which employed matrix multiplication to secure data, working on blocks of binary-coded plaintext. While effective for classical data, it lacked the state transformations present in our approach. As demonstrate before in [6] to enchant the encryption to be more robust, permutation codes can be applied to enhance data security by rearranging the positions of data bits based on a specific permutation pattern, creating a layer of obfuscation. Permutation matrices are commonly employed in constructing permutation codes. They are binary matrices with exactly one entry of 1 in each row and column and zeros elsewhere, representing a reordering of vector components. Here, we want to create a new matrix key in each block so that the encryption key for each block is different for each other. We can implement this method by combining the concept of Hill cipher techniques and permutation codes to allows for the creation of more robust encryption mechanisms. The method involving to represent the code as a matrix form as explained in[7].

As in [12] explored permutation ciphers applied to binary data, emphasizing reordering to conceal information. In contrast, our method advances this concept by integrating permutations with unitary matrices, creating a secure system for 8-qubits-encoded information that remains balanced and retains properties essential for decoding without data loss.

Additionally, in [12], the combination of different ciphers to bolster data protection was highlighted, often involving hybrid block and stream techniques [16] to thwart cryptanalysis. Our framework achieves similar robustness by generating unitary matrices iteratively (e.g., $Un = f(U_{n-1})$, enhancing each encryption cycle. This produces a distinct key for each block of binary data. This layered approach ensures decoding can only be achieved through the correct application of each key in each block, maintaining data security in binary code systems.

In this paper will explain an approach that adapts these classical methods for encoding information within 8-qubits quantum systems, specifically leveraging unitary matrix and permutation transformation to represent and manipulate qubit states in matrix form. The purpose of our research is to explore and enhance the application of binary coding theory in encryption and decryption mechanisms by employing matrix-based methodologies within an 8-qubit system. We will integrate the unitary matrix U as a primary component within the Hill cipher framework, combined with a permutation matrix P, forming the encryption formula $X' = X \cdot U \cdot P$. Additionally, we aim to extend this approach to recursive encryption by constructing a sequence of blocks, where U_n serves as the key for block n. Our research will apply this advanced matrix-based structure to an 8-qubit system, ensuring that decryption is well-defined, thereby demonstrating the robustness of this approach for maintaining data integrity and enhancing cryptographic security.

As in quantum computing, unitary matrices [18] play a crucial role in ensuring the correct evolution and transformation of quantum states. As explained is [8] [10] [17] A unitary matrix U is a complex matrix that satisfies the condition $UU^\dagger = U^\dagger U = I$. $U^\dagger$ is the conjugate transpose of U and I is the identity matrix. This property guarantees the preservation of the norm of vectors, which is vital for maintaining the probabilistic interpretation of quantum states during computations. Unitary matrices are fundamental in quantum mechanics because they describe quantum gates that perform reversible transformations on qubits. These transformations are essential for the accurate manipulation of quantum information without losing coherence. For instance, in [3] [4] the Hadamard gate, represented by a unitary matrix, creates superpositions, a basic requirement in many quantum algorithms. The norm-preserving property of unitary matrices ensures that the probabilities of all possible outcomes in a quantum measurement always sum to one, reflecting the conservation of quantum information. This reversibility is also crucial as it allows any quantum operation to be undone, a necessary feature in complex quantum computations and algorithms.

Unitary matrices are indispensable in quantum algorithms such as Shor's algorithm for factoring integers [2]. These algorithms rely on unitary transformations to manipulate qubits and perform computations exponentially faster than classical algorithms. Additionally, unitary matrices are central to quantum error correction codes, as in [4] which protect quantum information against decoherence and other forms of quantum noise. These codes use unitary operations to encode and decode quantum information, allowing errors to be detected and corrected, thus preserving the integrity of the quantum computation.

The purpose of our research :

The purpose of our research is to delve deeply into and advance the application of binary coding theory in encryption and decryption processes by utilizing matrix-based methodologies within an 8-qubit quantum computing system. This exploration is motivated by the need to develop more robust, secure, and efficient cryptographic methods that can leverage the unique properties of quantum mechanics.

We propose the integration of the unitary matrix U as a fundamental component within the Hill cipher framework. The Hill cipher, a classical encryption method, typically involves matrix multiplication to transform blocks of plaintext into ciphertext. By using the unitary matrix, which in quantum computing refers to unitary matrices as a transformation of multiple-qubits states, we adapt this classical method to the quantum domain. Unitary matrices are essential in quantum computing because they preserve the norm of the quantum state, ensuring that the quantum system remains in a valid state throughout the computation process.

To further enhance the security of our encryption method, we combine the unitary matrix U with a permutation matrix P. The permutation matrix P reorders the elements of the vector it multiplies, adding an additional layer of complexity and security to the encryption process as demonstrate in [17]. We will show that the resulting encryption model can be expressed as $X' = X \cdot U \cdot PX' = X . U . P$, where X is the initial quantum state, and X' is the encrypted state.

One of the key innovations of our research is the introduction of a recursive structure for the encryption process. This involves creating a sequence of encryption blocks, with each block say block n having its own unique key matrix U_n. This iterative approach enhances security by ensuring that each block of data is encrypted with a different transformation, making it significantly more difficult for an adversary to decipher the entire dataset if they manage to decrypt one block. We will show how to do this recursive so that that the encryption for block n can be defined as $X_n{}' = X_n \cdot U_n \cdot P$, where n = 1, 2, 3,…. For instance, encoding an 8-bit message X into binary and encrypting it involves transforming it with U and scrambling it with P. Decryption shows that applying P^T followed by $U^\dagger$ retrieves X, demonstrating consistency with binary code theory principles while achieving enhanced data protection.

Our research also focuses on ensuring that the decryption process is well-defined and reliable. The decryption must correctly reverse the transformations applied during encryption to retrieve the original data without loss of information. This involves applying the inverse of the permutation matrix P, that in [14] can be treated as P^T, and the conjugate transpose of the unitary matrix U, denoted as $U^\dagger$. Thus, the decryption process can be described as $X = X' \cdot P^t \cdot U^\dagger$.

Given the properties of the 8-qubit system, we illustrate that a Hadamard matrix [9] can be effectively used as the unitary matrix U. The Hadamard matrix is widely used in quantum computing for creating superpositions, which are a foundational aspect of quantum computation. By integrating the Hadamard matrix into our encryption sequence, we can leverage its ability to transform quantum states in a way that maximizes their informational entropy, further enhancing the security of the encrypted data.

Through practical examples, we demonstrate how our method integrates the Hadamard matrix for the encryption sequence. We provide detailed steps and implementations in Python to verify and explain the outcomes, ensuring that the theoretical underpinnings of our approach are grounded in practical, reproducible results.

Our paper showed the robustness of this combined method for maintaining data integrity and enhancing cryptographic security. By leveraging the principles of binary coding theory and quantum mechanics, we aspire to create a sophisticated encryption framework that is resilient against contemporary cryptographic attacks and poised to withstand future developments in quantum computing

Then as described above, the main purpose of our research is to introduce an advance the application of binary coding theory in encryption and decryption processes to an 8-qubits system of quantum computing by we combine the unitary matrix U with a permutation matrix P and we will show that the resulting encryption model can be expressed as $X' = X \cdot U \cdot PX' = X . U . P$, where X is the initial quantum state, and X' is the encrypted state. Furthermore, a recursive structure for the encryption process can be made. We will show how to do this recursive so that that the encryption for block n can be defined as $X_n{}' = X_n \cdot U_n \cdot P$, where $n = 1, 2, 3, ...$ This involves creating a sequence of encryption blocks, with each block say block n having its own unique key matrix U_n. This iterative approach enhances security by ensuring that each block of data is encrypted with a different transformation, making it significantly more difficult for an adversary to decipher the entire dataset if they manage to decrypt one block. Furthermore, we will create an example of this unitary matrix using the Hadamard matrix, as described in [20], and implemented this in Python as an example. The resulting program encrypts and decrypts using the Hadamard matrix as the unitary matrix and random permutations as the keys. This practical implementation demonstrates the feasibility and effectiveness of our proposed method, providing a robust solution for secure quantum state encryption.

II.METHOD

In classical computing, each bit in a binary sequence corresponds directly to a single qubit when represented in quantum form [5]. Each bit of the classical binary representation can be represented by a qubit. For the letter 'A', which is 01000001, we would need an 8-qubit register where each qubit represents one bit of the binary string. If a quantum algorithm required processing multiple letters or data, qubits could be set in a superposition state such as $|\psi\rangle = \alpha|01000001\rangle + \beta|01000010\rangle + \gamma|01000011\rangle$. With the asumtion that $\alpha^2 + \beta^2 + \gamma^2 = 1$ [4].

To fully utilize quantum operations and encryption methods, it is essential to represent the state of an 8-qubit system in matrix form. An 8-qubit system can be described as a state vector in a 256-dimensional complex vector space, given that there are $2^8 = 256$ possible combinations of binary states for the qubits [4]. This representation allows for advanced operations, such as linear transformations as in [8], which are foundational for quantum computation and encryption algorithms. Each basis state $i\rangle$ in an 8-qubit system corresponds to a unique combination of eight bits, where i ranges from 0 to 255. To represent the state $|\Psi\rangle$ in matrix form, we construct a column vector of size 256×1. Each element of this vector corresponds to one of the basis states and its associated probability amplitude. In general, the state vector could be written as:

$$|\Psi\rangle = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{254} \\ c_{255} \end{bmatrix}.$$

Our result :

In this paper we will show how do an encryption with a unitary transformation to a quantum state to securely encode information. To achieve this, we need to define a 256×256 unitary matrix U. A unitary matrix U is a complex matrix that satisfies the condition $UU^\dagger = U^\dagger U = I$. $U^\dagger$ is the conjugate transpose of U and I is the identity matrix [8]. This is obvious from the definition that $U^{-1} = U^\dagger$. Using the unitary matrix U, we will perform the encryption while preserving the normalization condition of the quantum state. This ensures that the total probability of all possible states remains 1.

In the next step, we will use a permutation matrix P. A permutation matrix P is a special type of square matrix used in various computational and mathematical applications to rearrange or permute the elements of a vector or the rows and columns of another matrix. Permutation matrices are binary matrices that have exactly one entry of 1 in each row and each column, with all other entries being 0. This structure ensures that when a permutation matrix is multiplied with another matrix or vector, it reorders the elements according to a specific permutation. Hence, the inverse of a permutation matrix P is the inverse of the permutation, that is a transpose of P denoted by $P^\top$.

This The primary objective of this study was to implement and evaluate the encryption and decryption of an 8-qubit system using unitary and permutation matrices. We will the show that the matrix UP remained unitary, allowing the creation of n blocks for n different 8-qubit states. The use of unique unitary matrices for each block prevented potential security vulnerabilities associated with using a single key for multiple blocks. The use of the a unitary matrix, combined with permutation matrices, provided a robust framework for secure quantum state encryption. Here we will analyze and show that the unitary matrix combined with permutation matrices provides a robust solution for quantum encryption of state vectors by proofing that the encryption matrix and the decryption matrix that was design as the key encrypted the original 8-qubits state X to new 8-qubits state X' and decrypted these 8-qubits state X' back to the original 8-qubits state X, adding an additional layer of security. The encryption is well defined if the resulting X' is indeed a 8-qubits state that is a state that has 1 as the cumulative total value of its constant in every 256 representations. We will show next that the combination of these matrices ensured that the state remained well-defined and normalized throughout the process. The iterative generation of new unitary matrices U_n from U_{n-1} for each block of 8-qubit states added an additional layer of security. This iterative method ensured that each block had a unique key, enhancing the overall security of the encryption process.

The decryption process, utilizing the inverse operations of the permutation and unitary matrices, demonstrated that the original quantum state could be accurately recovered. This result validates the encryption method, as the integrity of the quantum information was preserved.

To perform the encryption, we will use the unitary matrix U and the permutation matrix P. The encryption process can be described by the following operation $X' = XUP$. In order for this operation to be well defined we need to ensure that the matrix UP is also a unitary matrix so that we can perform the encryption while preserving the normalization condition of the quantum state. To proof this we need to proof that the matrix UP satisfies $(UP)(UP)^{\dagger} = I$. Since P is a permutation matrix and the inverse of P is its transpose then we have that $P^{\dagger} = P^{\mathsf{T}}$. So that $(UP)(UP)^{\dagger} = U.P.P^{\dagger}U^{\dagger} = I$. This complete our proof and ensuring that the matrix UP is indeed a unitary matrix.

Next, we would like to encrypt any state of an 8-qubit system using the matrix UP, where U is an arbitrary unitary matrix and P is an arbitrary permutation matrix. Suppose we have an 8-qubit system in a specific quantum state, and the matrix representation of this state is denoted by X. Let U be an arbitrary unitary matrix and P be an arbitrary permutation matrix. To encode matrix X, we will perform the following computation, resulting in the encrypted state $X' = X \cdot U \cdot P$. This operation ensures that the encoded state X' is a valid representation of an 8-qubit state. We have previously demonstrated that this matrix X' is indeed well-defined and accurately represents the state of the 8-qubit system. In other hand we can also decrypted X' to X by using the formula $X = X' \cdot P^T \cdot U^{\dagger}$.

Furthermore, the previous proof allows us to create a new unitary matrix UP using unitary matrix U and permutation matrix P. Reapplying this method, we can also create a unitary matrix UPP. Here, for each block say for example, block n, we have unitary matrix $UPP \dots P$ with the number of matrices P equal to n. This allows as to make an iteration with the formula as follow, $U_n = U_{n-1}$. In summarize the encryption of block n of the sequence of 8-qubits state is $X_n' = X_n \cdot U_n \cdot P$ and the decryption of block n of the sequence of 8-qubits state is $X_n = X_n' \cdot P^T \cdot U_n{}^{\dagger}$.

successfully implemented and evaluated an encryption and decryption method for an 8-qubit system using unitary and permutation matrices. The integration of these matrices ensured that the encryption process remained robust, with the matrix UP retaining its unitary properties, allowing for the creation of multiple encryption blocks. By using unique unitary matrices for each block, the approach effectively mitigated security risks associated with reusing a single key. This combination of unitary and permutation matrices provided a well-defined and normalized quantum state throughout the encryption process. The iterative generation of new unitary matrices U_n for each block added an additional layer of security, ensuring each block had a distinct key. The decryption process, employing the inverse operations of the permutation and unitary matrices, accurately recovered the original quantum state, validating the encryption method and preserving the integrity of the quantum information. This demonstrates the robustness and effectiveness of the proposed quantum encryption framework.

Compariosn to the previus result :

Please note that the encryption/decryption scheme we referenced can theoretically be implemented on modern quantum hardware. However, verifying operations involving hundreds of qubits would require rigorous calibration and error correction, which is beyond the scope of our research. Our focus is on the theoretical foundation, particularly in coding theory, and we adapt classical methods for quantum computing. Specifically, we work with an 8-quantum-state approach. The comparison lies in our encryption method, which is more secure because it employs a double-layer encryption scheme. This method is indeed more robust than single-layer encryption.

Implementation on Python :

To summarized this write the algorithm as follows.

Algorithms 1 showed how this method works using a random unitary matrix.

```
Algorithm 1
    Step 1: Initialization
    • INPUT: State vector $X$ (256x1)
    • DEFINE: Unitary matrix $U$ (256x256)
    • DEFINE: Permutation matrix $P$ (256x256)
    Step 2: Encryption
FUNCTION Encrypt($X, U, P$)
$X' = X \times U \times P$
RETURN $X'$
    Step 3: Decryption
FUNCTION Decrypt($X', U, P$)
$U^\dagger$ = ConjugateTranspose($U$)
$P^T$ = Transpose($P$)
$X = X' \times P^T \times U^\dagger$
RETURN $X$
    Main Program
$X'$ = Encrypt($X, U, P$)
ORIGINAL$_X$ = Decrypt($X', U, P$)
    Verify
IF ORIGINAL$_X \approx X$ THEN
PRINT "Decryption successful."
ELSE
PRINT "Decryption failed."
```

Algorithms 2 showed how this method works using the iteration process as defined above so that for each block we generate a new U_n from the previous U_{n-1} resulting in a new key for the block n. The recursive proses are done by the matrix product of unitary U_{n-1} and permutation matrix P resulting in a more secure encryption proses than algorithm 1.

```
Algorithm 2
Initialization
Input: State vector $X$ (256x1)
num_qubits = 8
$U_0$ = Hadamard matrix normalized by $\sqrt{256}$
$P$ = create_permutation_matrix(256)

Function create_unitary_matrices(num_qubits):
$U = U_0$
For $i$ from 1 to num_qubits - 1:
$U = U \cdot P$
Return $U, P$

Function encrypt(state, unitary, permutation):
$X' = state \cdot unitary \cdot permutation$
Return $X'$

Function decrypt(encrypted_state, unitary, permutation):
$U_{inv} = unitary^T$
$P_{inv} = permutation^T$
$X = encrypted\_state \cdot P_{inv} \cdot U_{inv}$
Return $X$

Main Program
original_state = random complex vector of size 256
Normalize original_state
unitary_matrix, permutation_matrix = create_unitary_matrices(num_qubits)
encrypted_state = encrypt(original_state, unitary_matrix, permutation_matrix
decrypted_state = decrypt(encrypted_state, unitary_matrix, permutation_matri
Verify if $decrypted\_state \approx original\_state$
```

To demonstrate how it works we will now try to write an example by writing a python program for this method. First, for an 8-qubits we represent this 8-qubits in matrix form X. Then we need an unitary matrix P. We will use a Hadamard matrix for this unitary matrix P as shown in [9] . The Hadamard matrix X for a single qubit is defined as :

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Creating a Hadamard matrix for 2^8 (which is 256) means constructing a 256×256 matrix. The Hadamard matrix can be recursively generated using the following formula:

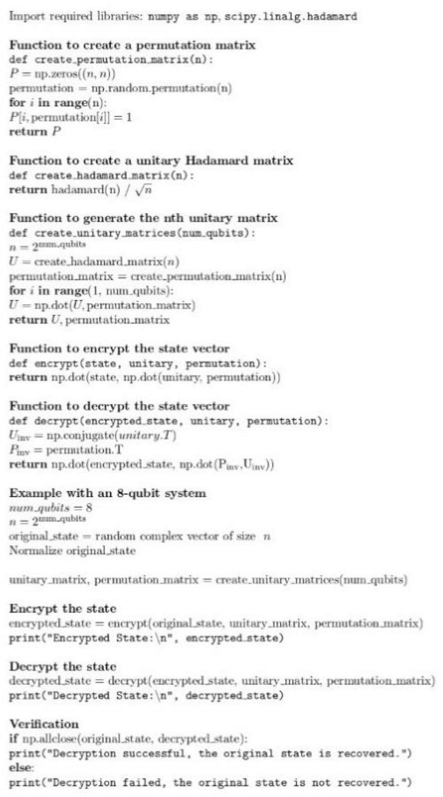
$$H_{2^n} = \frac{1}{\sqrt{2}}\begin{pmatrix} H_{2^{n-1}} & H_{2^{n-1}} \\ H_{2^{n-1}} & -H_{2^{n-1}} \end{pmatrix}$$

For $n = 8$ the resulting Hadamard matrix H_{256} is a matrix of size 256×256. In ython you can use the Hadamard library to create a Hadamard matrix.

The final python project will be created as follows

```
import numpy as np
from scipy.linalg import hadamard

Function to create a permutation matrix:
        def create_permutation_matrix(n):
        P = np.zeros((n, n))
        permutation = np.random.permutation(n)
        for i in range(n):
        P[i, permutation[i]] = 1
        return P
Function to create a unitary Hadamard matrix for 8 qubits:
        def create_hadamard_matrix(n):
        return hadamard(n) / np.sqrt(n)
Function to encrypt the state vector:
        def encrypt(state, unitary, permutation):
        return np.dot(state, np.dot(unitary, permutation))
Function to decrypt the state vector:
        def decrypt(encrypted_state, unitary, permutation):
        unitary_inv = np.conjugate(unitary.T)
        permutation_inv = permutation.T
        return np.dot(encrypted_state, np.dot(permutation_inv, unitary_inv))
Example with an 8-qubit system (n = 256):
        n = 256
        original_state = np.random.rand(n) + 1j * np.random.rand(n)
        original_state /= np.linalg.norm(original_state)  # Normalize the state

        unitary_matrix = create_hadamard_matrix(n)
        permutation_matrix = create_permutation_matrix(n)

        # Encrypt the state
        encrypted_state = encrypt(original_state, unitary_matrix, permutation_matrix)
        print("Encrypted State:\n", encrypted_state)

        # Decrypt the state
        decrypted_state = decrypt(encrypted_state, unitary_matrix, permutation_matrix)
        print("Decrypted State:\n", decrypted_state)

        # Verify that the decrypted state is the same as the original state
        if np.allclose(original_state, decrypted_state):
        print("Decryption successful, the original state is recovered.")
        else:
        print("Decryption failed, the original state is not recovered.")
```

And the Python program to run the iteration as defined above are as follows



```
Import required libraries: numpy as np, scipy.linalg.hadamard

Function to create a permutation matrix
def create_permutation_matrix(n):
P = np.zeros((n, n))
permutation = np.random.permutation(n)
for i in range(n):
P[i, permutation[i]] = 1
return P

Function to create a unitary Hadamard matrix
def create_hadamard_matrix(n):
return hadamard(n) / √n

Function to generate the nth unitary matrix
def create_unitary_matrices(num_qubits):
n = 2^num_qubits
U = create_hadamard_matrix(n)
permutation_matrix = create_permutation_matrix(n)
for i in range(1, num_qubits):
U = np.dot(U, permutation_matrix)
return U, permutation_matrix

Function to encrypt the state vector
def encrypt(state, unitary, permutation):
return np.dot(state, np.dot(unitary, permutation))

Function to decrypt the state vector
def decrypt(encrypted_state, unitary, permutation):
U_inv = np.conjugate(unitary.T)
P_inv = permutation.T
return np.dot(encrypted_state, np.dot(P_inv, U_inv))

Example with an 8-qubit system
num_qubits = 8
n = 2^num_qubits
original_state = random complex vector of size  n
Normalize original_state

unitary_matrix, permutation_matrix = create_unitary_matrices(num_qubits)

Encrypt the state
encrypted_state = encrypt(original_state, unitary_matrix, permutation_matrix)
print("Encrypted State:\n", encrypted_state)

Decrypt the state
decrypted_state = decrypt(encrypted_state, unitary_matrix, permutation_matrix)
print("Decrypted State:\n", decrypted_state)

Verification
if np.allclose(original_state, decrypted_state):
print("Decryption successful, the original state is recovered.")
else:
print("Decryption failed, the original state is not recovered.")
```

The Python code leverages the libraries numpy and scipy to perform complex matrix operations essential for quantum state manipulation. The numpy library is utilized for its powerful numerical operations, enabling efficient handling of matrices and vectors, while scipy's hadamard function provides the Hadamard matrix, known for its orthogonal properties crucial in quantum computations.

The function create_permutation_matrix(n) generates an $n \times n$ permutation matrix. This matrix is constructed by shuffling the rows such that each row and column contains exactly one "1", ensuring that the matrix effectively reorders elements during multiplication. The function begins by initializing an $n \times n$ zero matrix P. It then creates a random permutation of integers from 0 to $n-1$ and populates the matrix P so that each row places a "1" in a column specified by the permutation.

The create_hadamard_matrix(n) function returns an $n \times n$ Hadamard matrix, normalized by $\frac{1}{\sqrt{n}}$. The normalization is essential in quantum computations to maintain the normalization of quantum states. The function utilizes scipy's hadamard(n) to generate the Hadamard matrix and then divides it by $\sqrt{n}$ to normalize it.

The create_unitary_matrices(num_qubits) function aims to generate a unitary matrix by iteratively multiplying a Hadamard matrix by permutation matrices, simulating a series of quantum operations. The function calculates $n = 2^{num\_qubits}$ to accommodate the 8-qubit system, resulting in a 256-dimensional state vector. It initializes the matrix U with the normalized Hadamard matrix and iteratively multiplies U by a permutation matrix $num_{qubits} - 1 \times$, ensuring the transformations are complex and effective. The function returns the final unitary matrix and the last permutation matrix used.

The encrypt(state, unitary, permutation) function encrypts the state vector by sequentially multiplying it with the unitary and permutation matrices, following the formula $X' = X \cdot U \cdot P$. The function returns the encrypted state vector. Conversely, the decrypt(encrypted_state, unitary, permutation) function decrypts the state by applying the inverse of the unitary and permutation matrices. The decryption process uses the formula $X = X' \cdot P^t \cdot U^t = X' \cdot P^{-1} \cdot U^{-1}$, where U^{-1} is the transpose conjugate of U and P^{-1} is both the transpose and inverse of P. This ensures the correct retrieval of the original state vector, maintaining the integrity of the quantum information.

In an example run, the code handles an 8-qubit system where $num\_qubits = 8$ results in $n = 256$, creating a 256-dimensional state vector. A complex random vector is generated and normalized as the original state. During encryption, the state vector is transformed using the generated unitary and permutation matrices. The decryption process then reverses these transformations, with the decrypted state being compared to the original using the np.allclose() function. This comparison confirms that the decrypted state is approximately equal to the original, validating the success of the encryption-decryption cycle.

The original output of this program are complex matrices 256×256 both in encrypted state and in decrypted state in which we needs a lot of space to visualize but we simplified the output of this program for a few firs and lastt term and displayed it as follows :

```
Encrypted State:
[2.34038526e-02+2.17509742e-02j  -6.03981802e-
03+4.56055155e-02j                1.69003759e-
02+8.12635722e-03j    ...    ...  -1.10377420e-
02+3.91224586e-02j                4.35575304e-
02+1.04051998e-03j]
Decrypted State:   [0.05437421+0.07096808j
0.04757264+0.07325192j  .03779137+0.05093413j
0.06995303+0.0141159j      ...     ...     …
0.06696016+0.01274167j
0.06921159+0.04156791j
0.01065823+0.01309069j]
Decryption successful, the original
state is recovered.
```

III.RESULT AND DISCUSSION

In classical cryptography, as described in references [5], [6], and [7], encryption and decryption methods are employed to secure binary data using various ciphers. The Hill Cipher and Permutation Method are prime examples of such techniques. However, these methods are considered relatively weak when used alone in modern contexts. To address this, it is essential to develop new methods or enhance existing ones by combining them to create stronger encryption and decryption systems. For instance, studies in [16] and [17] introduced novel encryption and decryption methods involving both block and stream ciphers, as explained in [5], to ensure robust data security.

The primary objective of this study was to implement and evaluate an encryption and decryption method for an 8-qubit system from quantum computing in which was introduced by Feynman's introduction of the quantum system in 1982 [1], similar to the approaches in [16] and [17]. We utilized unitary and permutation matrices for encryption, drawing inspiration from [18] to preserve the unitary of the 8-qubit quantum state, ensuring that the transformation results remained valid quantum states [4]. Our results indicate that the matrix UP, where U is the unitary matrix and P is the permutation matrix, remained unitary. This facilitated the creation of n blocks for n different 8-qubit states. This finding is significant because it confirms that the unitary nature of the encryption matrix is preserved, an essential requirement for quantum computing operations.

By using unique unitary matrices for each block, we effectively mitigated potential security vulnerabilities associated with employing a single key for multiple blocks. This approach aligns with contemporary cryptographic practices that emphasize key uniqueness to prevent cross-block security breaches, as outlined in [5]. The combination of unitary and permutation matrices established a robust framework for quantum state encryption. This dual-matrix method ensures that state vectors are encrypted securely, adding an extra layer of protection that is crucial for maintaining data integrity in quantum communication systems.

Our methodology confirmed that the combined use of unitary and permutation matrices ensures the encrypted state remains well-defined and normalized throughout the process. This is critical for quantum systems, as maintaining normalization prevents loss or distortion of quantum information. The iterative generation of new unitary matrices U_n from U_{n-1} for each block of 8-qubit states added an additional layer of security. Each block having a unique key significantly enhances the overall security of the encryption process, a feature that is vital for scalable quantum encryption systems. The key for each of this block can be calculate using the formula $U_1 = U$ and $U_n = U_{n-1}P$. Consequently, the encryption process is achieved by applying the following operations $X' = X \cdot U \cdot P$ and the decryption process is achieved by applying the following operations $X = X' \cdot P^T \cdot U^\dagger$. Where X is the original state vector, X is the unitary matrix, and P is the permutation matrix.

Furthermore, we successfully created an example of this unitary matrix using the Hadamard matrix, as described in [21], and implemented this in Python as an example. The resulting program encrypts and decrypts using the Hadamard matrix as the unitary matrix and random permutations as the keys. This practical implementation demonstrates the feasibility and effectiveness of our proposed method, providing a robust solution for secure quantum state encryption.

These findings highlight the importance of combining unitary and permutation matrices to develop secure quantum encryption methods in 8-qubits system, ensuring that quantum states remain well-defined and normalized throughout the process. This approach not only enhances security but also provides a scalable solution for larger quantum systems, paving the way for further advancements in quantum cryptography and secure communication protocols.

IV.CONCLUSION

In conclusion, as in our main objective, this paper successfully implemented and evaluated an encryption and decryption method for an 8-qubit system introduce an advance the application of binary coding theory in encryption and decryption processes to an 8-qubits system of quantum computing by using unitary and permutation matrices. The integration of these matrices ensured that the encryption process remained robust, with the matrix UP retaining its unitary properties, allowing for the creation of multiple encryption blocks using the formula $U_n = U_{n-1}$ and $U_1 = U$. we showed that the resulting encryption model can be expressed as $X' = X \cdot U \cdot PX' = X \,.\, U \,.\, P$, where X is the initial quantum state, and X' is the encrypted state. The iterative generation of new unitary matrices U_n for each block added an additional layer of security, ensuring each block had a distinct key.. We also successfully created an example of the unitary matrix using the Hadamard matrix. The resulting program encrypts and decrypts using the Hadamard matrix and random permutations as the keys. This practical implementation demonstrates the existence of our proposed method.

REFERENCE

[1] R. P. Feynman, "Simulating physics with computers," International Journal of Theoretical Physics, vol. 21, p. 467–488, 1982. DOI : 10.1007/BF02650179

[2] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," IEEE Symposium on Foundations of Computer Science (FOCS), vol. 35, p. 124–134, 1994.

[3] M. C. I. Nielsen, Quantum Computation and Quantum Information 10th Anniversary Edition, New York, USA: Cambridge University Press, 2010.

[4] R. LaPierre, Introduction to Quantum, Springer, 2021.

[5] W. Stallings, Cryptography and Network Security 7th edition, Pearson, 2017.

[6] O. Goldreich, Foundations of CryptographyBasic Tools, Cambridge University Press, 2004.

[7] S. X. C. Ling, Coding Theory, Cambridge University Press, 2004.

[8] S. Roman, Advanced Linear Algebra, USA: Springer Science+Business Media, LLC, 2008.

[9] C. B.J., “Hadamard Matrix and its Application in Coding,” International Journal of Mathematics Trends and Technology (IJMTT), vol. 59, no. 4, pp. 218-227, 2018. DOI : /10.14445/22315373/IJMTT-V59P532

[10] A. Horn, Matrix Analysis, Cambridge University Press, 2012.

[11] Lester S. Hill, Cryptography in an Algebraic Alphabet, *The American Mathematical Monthly* Vol.36, pp. 306–312, June–July 1929. DOI : 10.2307/2298294

[12] Lester S. Hill, Concerning Certain Linear Transformation Apparatus of Cryptography, *The American Mathematical Monthly* Vol.38, pp. 135–154, 1931. DOI : 10.2307/2300969

[13] Jeffrey Overbey, William Traves, and Jerzy Wojdylo, On the Keyspace of the Hill Cipher, *Cryptologia*, Vol.29, No.1, pp59–72, 2005 DOI : 10.1080/0161-110591893771

[14] Savard, John. "Methods of Transposition". A Cryptographic Compendium. Retrieved 27 June 2023.

[15] Shi, Z., Lee, R.B., Implementation complexity of bit permutation instructions, Conference Record of the Thirty-Seventh Asilomar Conference on Conference: Signals, Systems and Computers Volume 1, 2003.

[16] Irene, G.Z. et al. Enhancing Image Security using Chaotic Map and Block Cipher, International Conference on Futuristic Innovations and Challenges to Diversity Management Emerging Technologies and Sustainability for Inclusive Industrial Growth, 2015.

[17] Führ., Hartmut; Rzeszotnik., Z,"A note on factoring unitary matrices". Linear Algebra and Its Applications. 547: 32–44. 2018. DOI :/10.1016/j.laa.2018.02.017

[18] Horodecki., R. et al.. Quantum entanglement. Reviews of Modern Physics. 81 (2): 865–942, 2009. DOI : 10.48550/arXiv.quant-ph/0702225

[19] Hedayat., A. and Wallis., W. D. Hadamard Matrices and Their Applications. The Annals of Statistics Vol. 6, No. 6, pp. 1184-1238. 1978. DOI : 10.1214/aos/1176344370