International
SMART Journal
International SOCIAL MENTALITY AND RESEARCHER THINKERS Journal
e-ISSN: 2630-631X

RESEARCH ARTICLE
MATHEMATİCS AND SCİENCE EDUCATİON

The Role of Planar Graphs in Encryption and Decryption in Secure Communications: An Example

Güvenli İletişimde Planar Grafların Şifreleme ve Çözmedeki Rolü: Bir Örnek

Yılmaz Gür [1]
Hülya Gür [2,3]

ABSTRACT

This paper presents an innovative algorithm for encryption and decryption in Python, using the properties of planar graphs to ensure secure data transmission. The proposed symmetric encryption method uses the concepts of cycle graphs, planar graphs, and minimal spanning trees to generate a complex ciphertext using a shared key. This approach improves the security and efficiency of the encryption process. By employing these graph-based structures, the algorithm generates intricate patterns that enhance the security of the encrypted data, making it resistant to unauthorized decryption. The use of cycle graphs contributes to creating complex, non-repetitive patterns, while planar graphs help in efficiently organizing data within a secure structure. The minimal spanning tree concept plays a critical role in improving the computational efficiency of the encryption and decryption operations. This novel approach not only strengthens security but also offers a more efficient alternative to traditional encryption techniques. The use of planar graphs is integral in organizing and structuring the encryption process, while minimal spanning trees improve computational efficiency for both encryption and decryption operations. This approach not only offers an important level of security but also introduces an efficient and scalable method for secure data encryption and decryption in Python).

Keywords: Encryption-Decryption, Planar Graph, Cycle Graph.

ÖZET

Bu makale, güvenli veri iletimi sağlamak için düzlemsel grafların özelliklerini kullanan yenilikçi bir şifreleme ve şifre çözme algoritması sunmaktadır. Önerilen simetrik şifreleme yöntemi, döngüsel graflar, düzlemsel graflar ve minimal spinning ağaç kavramlarını kullanarak paylaşılan bir anahtar ile karmaşık bir şifreli metin oluşturur. Bu yaklaşım, şifreleme sürecinin güvenliğini ve verimliliğini artırır. Bu graf tabanlı yapıları kullanarak, algoritma şifreli verilerin güvenliğini artıran karmaşık desenler oluşturur ve yetkisiz şifre çözmeye karşı dirençli hale getirir. Döngüsel graflar, karmaşık ve tekrarsız desenler oluşturulmasına katkı sağlarken, düzlemsel graflar, verilerin güvenli bir yapıya etkin bir şekilde organize edilmesine yardımcı olur. Minimal spinning ağaç kavramı, şifreleme ve şifre çözme işlemlerinin hesaplama verimliliğini artırmada önemli bir rol oynar. Bu yenilikçi yaklaşım, sadece güvenliği güçlendirmekle kalmaz, aynı zamanda geleneksel şifreleme tekniklerine göre daha verimli bir alternatif sunar. Düzlemsel graflar, şifreleme sürecini organize etme ve yapılandırma konusunda önemli bir rol oynarken, minimal spinning ağaçlar, hem şifreleme hem de şifre çözme işlemleri için hesaplama verimliliğini artırır. Bu yaklaşım, yalnızca önemli bir güvenlik seviyesi sunmakla kalmaz, aynı zamanda Python'da güvenli veri şifreleme ve şifre çözme için verimli ve ölçeklenebilir bir yöntem sunar.

Anahtar Kelimeler: Şifreleme-Şifre Çözme, Düzlemsel Graf, Döngüsel Graf.

How to Cite This Article
Gür, Y. & Gür, H. (2025). "The Role of Planar Graphs in Encryption and Decryption in Secure Communications: An Example", International Social Mentality and Researcher Thinkers Journal, (Issn:2630-631X) 11(3): 428-436. DOI: https://doi.org/10.5281/zenodo.15544510

Arrival: 18 March 2025
Published: 30 May 2025

INTRODUCTION

In this article, we consider an undirected graph G (V, E), where V denotes the set of vertices and E denotes the set of edges connecting pairs of vertices. A path in this graph is defined as a sequence of vertices connected by edges, with the constraint that no vertex appears more than once in the sequence. Thus, a path is a traversal from one vertex to another, where the uniqueness of each vertex along the route is preserved. A cycle is a particular type of path in which a traversal starts from one vertex, passes through the other vertices, and eventually returns to the same starting vertex. This form of traversal forms a closed loop, called a cycle. A cycle graph is a graph that has one or more such cycles and characterized by a structure in which the cycle includes all vertices in the graph and provides a complete and closed connection between the vertices.

Graph representations in computer science use two primary data structures: the adjacency list and the adjacency matrix. An adjacency list is a space-saving data structure consisting of an array of linked lists. Each

[1] Assoc. Prof. Dr., International University of Sarajevo, Faculty of Engineering and Natural Sciences, Department of Mechanical Engineering, Sarajevo, Bosnia and Herzegovina.
[2] Prof. Dr., International University of Sarajevo, Faculty of Education, Computer Education and Instructional Technology Department, Sarajevo, Bosnia and Herzegovina.
[3] Prof. Dr., Balıkesir University, Necatibey Educational Faculty, Mathematics and Science Education Department, Balıkesir, Türkiye.

index of the array corresponds to a vertex in the graph, and the linked list in each index has the addresses or references of the vertices next to that vertex. This structure is particularly useful for sparse graphs, where the number of edges is much smaller than the number of connections. On the other hand, the adjacency matrix is a two-dimensional array of size v, where v is the number of vertices in the graph. Each element of this matrix is the presence or absence of an edge between a pair of vertices. If an edge exists between vertex i and vertex j, the matrix entry a[i][j] will have a nonzero value (1); otherwise, it will have zero (0). The adjacency matrix is preferred for dense graphs, where the number of edges is large related to the number of vertices.

Cryptography is the science and practice of protecting communications from third-party interception, ensuring that only the intended recipient and sender can understand the content of a transmitted message Indhu, &Rekha, 2022). It involves applying mathematical algorithms to transform the original message (or plaintext) into a form that cannot be read by unauthorized people (Yamuna, et al. 2012). This transformation process, often called encryption, uses a cryptographic key to make the text unintelligible without the corresponding decryption key (Kumari&Kirubanad, 2018; Wardak, et al. 2024).

Cryptography can take forms, one of which involves replacing characters or symbols in the plaintext with other characters, a process called substitution encryption. The modified message that no longer resembles its original form is called ciphertext. This ciphertext can only be converted back to its original form (plaintext) by a person or system that has the correct decryption key. Therefore, cryptographic techniques are central to secure communication and data protection and form the basis of security protocols used in modern digital systems, such as email encryption, secure web browsing (https), and data storage security (Geetha&Regavi, 2022; Mohamed, 2020).

This paper introduces an innovative algorithm for encryption and decryption in Python and uses the mathematical properties of planar graphs to ease secure data transmission. The proposed symmetric encryption method combines the basic concepts of cyclic graphs, planar graphs, and minimal spanning trees to generate a robust and complex ciphertext using a shared key (see Appendix). Using these graph-based structures, the algorithm creates complex patterns that significantly increase the security of encrypted data and make it resistant to unauthorized decryption.

The algorithm uses a cyclic graph to create complex, non-repetitive patterns that are difficult to solve without the key. The use of planar graphs offers an effective way to structure and organize encrypted data, making the encryption process both secure and computationally efficient. Additionally, minimum spanning trees used to improve the computational complexity of both encryption and decryption operations, reducing the time and resources needed for secure communication (Kumari, M&Kirubanad, 2018).

This novel approach not only strengthens the security of the encryption process but also provides an efficient and scalable method for implementing encryption and decryption in Python. By exploiting the properties of graph theory, the proposed algorithm offers an important level of security and efficiency, making it a promising alternative to traditional cryptographic techniques.

METHODOLOGY

The encryption algorithm are given in the Table 1step by step.

Table 1: The encryption algorithm

Step	Description
1. Start Character	Add an individual character (e.g. A) to show the starting character of the plaintext.
2. Vertex Creation	Create a vertex for each character in the plaintext.
3. Edge Creation	Add edges between consecutive vertices (characters) to form a cycle graph.
4. Edge Weight Assignment	The weight of the edges is the distance between the vertices. Find the weights using the values in the coding table.
5. Planar Graph Construction	Add more edges to the graph to form a planar graph. Assign sequential weights starting from the maximum weight in the encoding table.
6. Minimum Spanning Tree (MST)	Compute the Minimum Spanning Tree (MST) using graph algorithms (e.g., Kruskal's or Prim's).
7. Matrix Representation	Store the vertices in a matrix, placing the vertex order along the diagonal.
8. Matrix Multiplication	Multiply the matrices to produce an intermediate result.
9. Shared-Key Encryption	Multiply the resulting matrix M_3 by the predefined shared-key K to form matrix C (ciphertext).
10. Ciphertext Construction	Create the ciphertext generated in linear format with matrix C and matrix M1 from step 8

The decryption algorithm are given in the Table 2.

Table 2: The encryption algorithm

Step	Description
1. Inverse Shared-Key	The receiver must compute M_3 which is the inverse of the shared-key K^{-1}.
2. Matrix Inversion	The receiver must compute M_2 which is the inverse of the matrix M_1 (ciphertext) using K^{-1}.
3. Original Text Recovery	Decode the resulting matrix M_1 using the encoding table to recover the original plaintext.

FINDINGS AND DISCUSSION

This chart succinctly summarizes both the encryption and decryption processes.

Example: Suppose we want to encrypt the plaintext 'BASE' to send it to the receiver. The first step in the encryption process is to convert the message into a graph. This is done by converting each character of the plain text into a corresponding vertex in the graph. For the message 'BASE', we create the following vertices: vertex v_1 for character B; vertex v_2 for A character; vertex v_3 for character S; vertex v_4 for character E. Thus, the graph starts with four vertices being each character of the plaintext.

Link each pair of sequential characters together to form a cycle graph, ensuring that the first and last characters are also connected to complete the cycle. Then, assign a weight to each edge using the encoding Table 3 provided below.

Table 3: Encoding table

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**
24	25	26																				
X	**Y**	**Z**																				

Then, the graph will be shown in Figure 1. This refers to the graph representation of the plaintext as a cycle graph Figure 2.

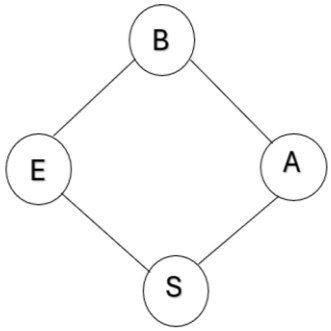

Figure 1: Connected Graph

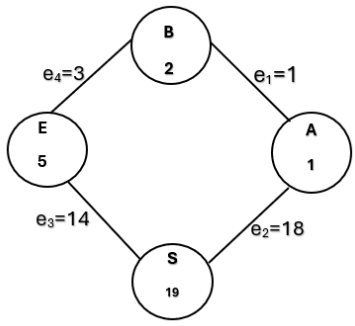

Figure 2: Wighted graph contained plain text characters

The edges are given:

$\mathbf{e_1} = |v_1\text{-}v_2| = |1\text{-}2| = 1,$ $\mathbf{e_2} = |v_2\text{-}v_3| = |1\text{-}19| = 18,$ $\mathbf{e_3} = |v_3\text{-}v_4| = |19\text{-}5| = 14,$ $\mathbf{e_4} = |v_4\text{-}v_5| = |5\text{-}2| = 3.$

This weighted graph can now be used in encryption algorithms (Figure 2). Each character is mapped to a vertex, and the connections between vertices (edges) have weights that can be used to define a more complex encryption process, such as when generating cipher text or using graph algorithms like minimum spanning trees.

A planar graph is defined as a graph that can be drawn on a plane without edges crossing. To form a planar graph, we need to add edges between the vertices (standing for the characters) in such a way that no two edges cross each other.

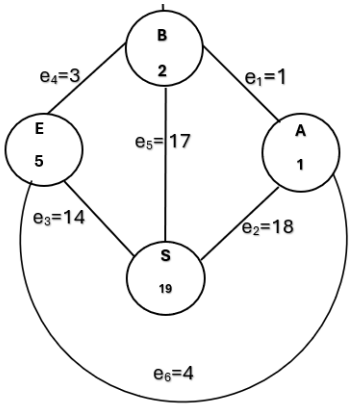

Figure 3: Planar graph

Once we have created the graph by adding all the vertices and edges, we can represent the graph as a matrix (specifically an adjacency matrix) (Figure 3). An adjacency matrix is a square matrix where: Each row and column corresponds to a vertex in the graph. If there is an edge between two vertices, the corresponding matrix cell has the weight of the edge between them. If there is no edge, the corresponding cell has 0 or infinity (depending on the approach).

When we continue adding edges to the planar graph, the coding table will be used for the weights of the edges. However, while calculating the weight of the newly added edge, a consecutive weight is added starting from the maximum weight in the coding table of each newly added edge. Special Character to Point to the First Character: To mark the start of the graph, we add a special character that points to the first character. This helps establish a reference point for encryption and decryption processes. Add a special character "#", "0" or "B" before the first character (e.g., "B"), which will point to the first character in the message (e.g., "A" ; would be # → X → A → S → ..... **)**(see Figure 4).

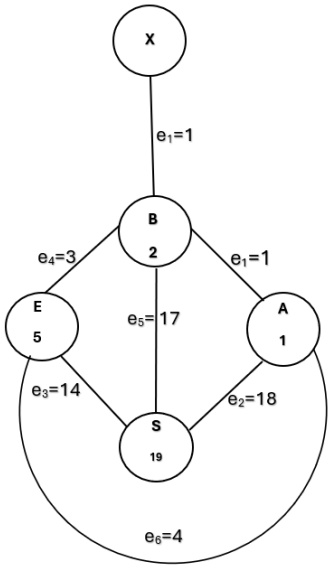

Figure 4: Planar graph with special character

$$[\boldsymbol{M_1}]_{5x5}=\begin{bmatrix}0 & 1 & 0 & 0 & 0\\ 1 & 0 & 1 & 17 & 3\\ 0 & 1 & 0 & 18 & 0\\ 0 & 17 & 18 & 0 & 14\\ 0 & 3 & 4 & 14 & 0\end{bmatrix}$$

We need to find the minumum spanning tree (Figure 5):

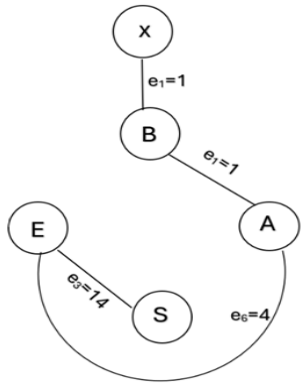

Figure 5: Planar Graph with special character

Minumum spanning tree matrix is given below:

$$\mathbf{M_2}= \begin{array}{c} \\ x \\ B \\ A \\ E \\ S \end{array} \begin{array}{c} \begin{array}{ccccc} x & B & A & E & S \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 & 14 \\ 0 & 0 & 0 & 14 & 0 \end{bmatrix} \end{array}$$

The number of characters is written in order in the elements on the diagonal of the adjacency matrix. Each character in the plaintext message is assigned a position or index based on its sequential appearance in the message. These indices, representing the relative order of the characters, are then placed in the diagonal elements of the adjacency matrix (Table 4).

Table 4: Character Values

Order number	**X**	**B**	**A**	**E**	**S**
Character	0	1	2	3	4

$$\mathbf{M_2}= \begin{array}{c} \\ x \\ B \\ A \\ E \\ S \end{array} \begin{array}{c} \begin{array}{ccccc} x & B & A & E & S \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 & 14 \\ 0 & 0 & 0 & 14 & 0 \end{bmatrix} \end{array}$$ and we change dioganal elements as seen new M_2 below:

$$\mathbf{M_2}= \begin{array}{c} \\ x \\ B \\ E \\ S \\ A \end{array} \begin{array}{c} \begin{array}{ccccc} x & B & A & E & S \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 4 & 0 \\ 0 & 0 & 4 & 3 & 14 \\ 0 & 0 & 0 & 14 & 4 \end{bmatrix} \end{array}$$

After that we multiply M_1 and M_2. Then $\mathbf{M_3= M_1.M_2}$.

$$\mathbf{M_3= M_1xM_2}= = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 17 & 3 \\ 0 & 1 & 0 & 18 & 0 \\ 0 & 17 & 18 & 0 & 14 \\ 0 & 3 & 4 & 14 & 0 \end{bmatrix} . \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 4 & 0 \\ 0 & 0 & 4 & 3 & 14 \\ 0 & 0 & 0 & 14 & 4 \end{bmatrix}$$ then

$$\mathbf{M_3} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 68 & 46 & 238 \\ 1 & 0 & 73 & 0 & 252 \\ 17 & 18 & 17 & 268 & 0 \\ 3 & 4 & 59 & 16 & 196 \end{bmatrix}$$

The upper diagonal matrix K is preferred to provide ease of calculation, efficiency, and reliability in encryption systems.

$$K = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{C = M_3 . K} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 68 & 46 & 238 \\ 1 & 0 & 73 & 0 & 252 \\ 17 & 18 & 17 & 268 & 0 \\ 3 & 4 & 59 & 16 & 196 \end{bmatrix} . \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

this is first chiper matrix:

$$C_1 = \begin{bmatrix} 22 & 24 & 218 & 330 & 686 \\ 21 & 24 & 217 & 330 & 686 \\ 21 & 22 & 149 & 284 & 448 \\ 20 & 22 & 76 & 284 & 196 \\ 3 & 4 & 59 & 16 & 196 \end{bmatrix}$$

The data to be send is:

22 24 218 330 686 21 24 217 330 686 21 22 149 284 448 20 22 76 284 196 3 4 59 16 196

Decryption Process:

The incoming code in the form of a 25-element row matrix in the example corresponds to a 5x5 matrix, and the K matrix size to be inverted is also 5x5.

The decryption process begins when the receiver receives the ciphertext. A ciphertext, denoted as C, is a transformed version of the original plaintext that has been encrypted using a shared encryption key K.

To retrieve the original plaintext, the receiver must reverse the encryption process using the appropriate decryption steps.

The first step in this process is to compute the matrix M_3, which is an intermediate matrix used for the decryption. This is achieved by multiplying the received ciphertext by the inverse of the shared key matrix, denoted as K^{-1}.

$$\mathbf{M_3 = C . K^{-1}}$$

Where: M_3 is the intermediate matrix that will contain the values used to reconstruct the original message. C is the received ciphertext. K^{-1} is the inverse of the shared key matrix K.

Multiplying the ciphertext by the inverse of the shared key matrix effectively undoes the transformations applied during encryption. The inverse of K ensures that the operations are reversed, restoring the original order and structure of the data.

Once M_3 is computed, the receiver can proceed with the next steps of the decryption process, which may involve additional matrix operations or the application of a decoding table, depending on the encryption method used.

$$\mathbf{K^{-1}} = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M_3 = C . K^{-1}} = \begin{bmatrix} 22 & 24 & 218 & 330 & 686 \\ 21 & 24 & 217 & 330 & 686 \\ 21 & 22 & 149 & 284 & 448 \\ 20 & 22 & 76 & 284 & 196 \\ 3 & 4 & 59 & 16 & 196 \end{bmatrix} . \begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M_3} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 68 & 46 & 238 \\ 1 & 0 & 73 & 0 & 252 \\ 17 & 18 & 17 & 268 & 0 \\ 3 & 4 & 59 & 16 & 196 \end{bmatrix}$$

Then calculate M_2 by multiplying M_3 by M_1^{-1}.

$$\mathbf{M_1} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 & 14 \\ 0 & 0 & 0 & 14 & 0 \end{bmatrix} \text{ and } \mathbf{M_2} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 2 & 4 & 0 \\ 0 & 0 & 4 & 3 & 14 \\ 0 & 0 & 0 & 14 & 4 \end{bmatrix} \text{ then}$$

$$\mathbf{M_1^{-1}} = \begin{bmatrix} 3 & 1 & -3/2 & -3/4 & 5/7 \\ 1 & 0 & 0 & 0 & 0 \\ -5/9 & 0 & -7/36 & 0 & 1/4 \\ -1/18 & 0 & 1/18 & 0 & 0 \\ -1/2 & 0 & 1/4 & 1/14 & -9/28 \end{bmatrix} \text{ and}$$

$$\mathbf{M_2^{-1}} = \begin{bmatrix} -31/54 & 1 & -23/54 & -1/27 & 7/54 \\ 1 & 0 & 0 & 0 & 0 \\ -23/54 & 0 & 23/54 & 1/27 & -7/54 \\ -1/27 & 0 & 1/27 & -1/54 & 7/108 \\ 7/54 & 0 & -7/54 & 7/108 & 5/216 \end{bmatrix}$$

$\mathbf{M_2 = M_1^{-1} . M_3}$

$$\mathbf{M_2} = \begin{bmatrix} 3 & 1 & -3/2 & -3/4 & 5/7 \\ 1 & 0 & 0 & 0 & 0 \\ -5/9 & 0 & -7/36 & 0 & 1/4 \\ -1/18 & 0 & 1/18 & 0 & 0 \\ -1/2 & 0 & 1/4 & 1/14 & -9/28 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 68 & 46 & 238 \\ 1 & 0 & 73 & 0 & 252 \\ 17 & 18 & 17 & 268 & 0 \\ 3 & 4 & 59 & 16 & 196 \end{bmatrix}$$

$$\mathbf{M_2} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 & 14 \\ 0 & 0 & 0 & 14 & 0 \end{bmatrix}$$ represent the following final graph (it is given below).

The matrix showed that Spanning tree (e_1-e_1-e_6-e_3) is a final graph as it can be sen in the Figure 5.

IMPLEMENTATIONS AND RESULTS

To implement and evaluate the proposed algorithm.The experimental tests were carried out using a variety of plain text inputs, which represented messages of different lengths. These tests were designed to assess the accuracy and efficiency of the algorithm. In the study, a random set of encryption keys was generated to test the performance of the algorithm, and the results were recorded for later analysis. Some of the key experimental outcomes were captured in Table 2, where the effect of different plaintext lengths and key sizes was explored. The findings obtained were considered from Experiments as key length flexibility; ciphertext growth; encryption time and algorithm efficiency with small plain texts. The algorithm's design allows it to work with public keys of arbitrary length. This is achieved by duplicating the key and extending it to te desired length. This flexibility ensures that the method can be adapted for varying security requirements, regardless of the key size. Case studies have shown that the size of the ciphertext is positively proportional to the length of the plaintext. As the plaintext message grows larger, the algorithm produces a larger ciphertext, which is expected in cryptographic systems. This suggests that the encryption algorithm is scalable; however, it is more efficient when applied to smaller plaintext messages. One notable observation from the experimental results is the increased time required for encryption as the size of the plaintext message grows. This is largely due to the computational complexity involved in the matrix multiplication procedure used in the encryption process. The algorithm's performance scales with the size of the input, and matrix multiplication, which forms the core of the encryption technique, adds to the time complexity. Consequently, while the algorithm is highly effective for smaller messages, it may require optimizations or alternative approaches for large-scale encryption tasks.

Despite the increase in ciphertext size and encryption time with larger plaintexts, the algorithm was found to be particularly efficient for smaller plaintexts (Paszkiewicz et al., 2001). This is a common trade-off in

encryption schemes, where smaller messages can be processed faster, allowing for quick encryption and decryption.

CONCLUSION FROM EXPERIMENTAL RESULTS

A strong aspect of the obtained results is that the proposed encryption algorithm effectively handles various plaintext sizes, is scalable, and flexible (Etaiwi, 2014; Sensarma&Sarma, 2014; Anandhi&Abarna, 2022; Indhu&Rekha, 2022). However, the time required for encryption increases with the size of the input, particularly due to the matrix multiplication involved. While this may not pose a problem for smaller messages, additional work may be necessary to improve the algorithm's performance for larger-scale applications. Further optimizations could be explored to reduce the computational overhead, particularly focusing on the matrix operations and potentially parallelizing the encryption process for more extensive datasets.In conclusion, while the algorithm shows promising results for small to medium-sized data, further optimizations are needed for broader use cases involving larger data volumes.

This paper explores a cryptographic algorithm that leverages planar graphs for secure data encryption and decryption. The approach uses the structural properties of planar graphs, such as their ability to have complex relationships between vertices and edges, to enhance the security of the encryption process. By employing the concepts of cycle graphs, minimal spanning trees, and weighted edges, the proposed algorithm offers a robust framework for encrypting messages in a manner that is resistant to unauthorized decryption (Indhu&Rekha, 2022).

The algorithm implemented in Phyton programming language, which provided a reliable and efficient platform for the algorithm's execution (Schneier,1996; Katz&Lindell, 2007). Phyton is chosen for its computational efficiency and flexibility in handling complex data structures, such as graphs and matrices, which are integral to the encryption process. All matrix operations (multiplication of matrices, inversion of matrices) were done using Python software codes (see Appendix). This algorithm evaluated and confirmed using different datasets, showing its functionality and potential for use in real-world applications.

In future work, the proposed algorithm could further develop and be implemented using other programming languages. Moreover, future improvements could focus on optimizing the algorithm for larger datasets, enhancing its scalability, and improving computational efficiency, especially in the context of large-scale encryption. With the continued advancements in programming languages and cryptographic research, there is significant potential for this algorithm to be refined and adapted for even more complex cryptographic tasks in the future.

REFERENCES

Anandhi, M., & Abarna, D. (2022). Encryption algorithm using graph theory. International Journal of Mechanical Engineering, 7(4), 853-859.

Schneier, B. (1996). Applied cryptography: Protocols, algorithms, and source code in C (2nd edition). Wiley.

Sensarma, D. & Sarma, S. S. (2014). GMDES: a graph based modified data encryption standard algorithm with enhanced security. International Journal of Research Engineering Technology, 3(3), 653-660.

Etaiwi, W. M. Al. (2014). Encryption algorithm using graph theory. Journal of Scientific Research and Reports, 3(19), 2519-2527.

Geetha, N. K. & Regavi, V. (2022). Graph theory matrix approach in cryptography and network security. 2022 Algorithms, Computing and Mathematics Conference (ACM), Chennai, India, 108-110.

Katz, J., & Lindell, Y. (2007). Introduction to modern cryptography (3rd edition). CRC Press.

Kumari, M., & Kirubanad, V. B. (2018). Data encryption and decryption using graph plotting. International Journal of Civil Engineering and Technology, 9, 36-46.

Mohamed, K. S. (2020). Introduction to Cyber Security. In New Frontiers in Cryptography 1-12. Springer, Cham.

Wardak, O., Sinha, D., & Sethi, A. (2024). Encryption and decryption of signed graph matrices through RSA algorithm. Indian Journal of Pure and Applied Mathematics, 55, 1477-1484.

Paszkiewicz, A., Górski, A. K., Gorski, K., Kotulski, Z. A., Kulesza, K., & Szczepanski, J. (2001). Proposals of graph-based ciphers, theory and implementations. ResearchGate, 1-10.

Yamuna, M., Gogia, M., Sikka, A., & Khan, J. H. (2012). Encryption using graph theory and linear algebra. International Journal of Computer Applications, 5(2), 102-107.

Indhu, K., & Rekha, S. (2022). An approach of graph theory on cryptography. Journal for Research in Applied Science and Engineering Technology, 10(11), 1587-1591.