1st International Conference on Computing & Artificial Intelligence

Fatima Business School

SALIM HABIB UNIVERSITY

DOCUMENTATION

Topic : Graph Theory-Based Cryptographic System Using Path-Encoding

COURSE INSTRUCTOR : Miss Meerab Amir

AUTHORS: Amna Mustafa Alwani - F24CSC048

Iqra Tariq - F24BSE009

Background

Cryptography is essential in protecting sensitive information in the digital era. It relies on complex mathematical principles to transform readable data into secure code. One such method integrates Graph Theory — a field of mathematics focused on the study of graphs, which are structures used to model pairwise relations between objects. This project utilizes fundamental concepts of Graph Theory to build an encryption-decryption mechanism that converts a message into a graph representation, ensuring security and integrity.

Objective

The primary objective of this project is to design and implement a cryptographic system that leverages principles of graph theory to ensure data confidentiality and integrity. By transforming plain text into graph-based structures, this system introduces an additional layer of complexity that significantly enhances encryption robustness. The process involves converting characters into binary form, manipulating them through graph-based algorithms, and embedding them into encrypted message graphs using custom keys. The system then accurately reverses this process for decryption, ensuring the original message is retrieved securely. This innovative approach aims to explore how graph theory can provide novel and effective strategies for securing communication in modern digital systems.

Program Description

This C program is made for encrypting message by using graph theory, especially hammilton graph. In addition, the encryption process also uses XOR Chipper. The decryption also uses the same theory as the encryption does.

The source code is at src folder, the abstract data type used (Graph, Key, etc) is in ADT folder, encryption program is in Encryption folder, and decryption program is in Decryption folder. The main program is to implement both encryption and decryption process, so user can use both at the same time.

Methodology

1. Text-to-Binary Conversion: Convert each character into a 7-bit binary value.
2. Bit Manipulation: Analyze binary arrays using functions like `countOne`, `countZero`, and `shuffle`.
3. Graph Construction: Represent the binary pattern as a directed or undirected graph.
4. KeyList Integration: Incorporate a key structure (keyList) that defines node connections and encoding strategy.
5. Encryption: Encode the message as a graph structure using the `msgToGraf()` and `encrypt()` functions.
6. Decryption: Reverse the process by using `grafToBin()` and `binToChar()` functions to recover the original message.

Implementation

The project is developed in C language, using modular code structure with separate files for encryption, decryption, and graph-related functions.

- **main.c:** Handles user input and invokes encryption and decryption functions.
- **encrypt.c / encrypt.h:** Contains functions for message encoding and graph construction.
- **decrypt.c / decrypt.h:** Handles graph-to-binary decoding and message recovery.
- **adt.c / adt.h:** Defines the Graph and KeyList data structures used across the project.

-boolean.h: Defines Boolean constants (TRUE, FALSE) and optionally a bool data type for logical operations throughout the program. This provides compatibility and clarity in conditions and logical expressions, especially in environments where <stdbool.h> may not be used.

HOW TO USE?

1. go to folder src

```
cd src
```

2. compile the main program (copy this command to your CLI)

```
gcc -o main main.c ./ADT/adt.c  ./Encrypt/encrypt.c ./Decrypt/decrypt.c
```

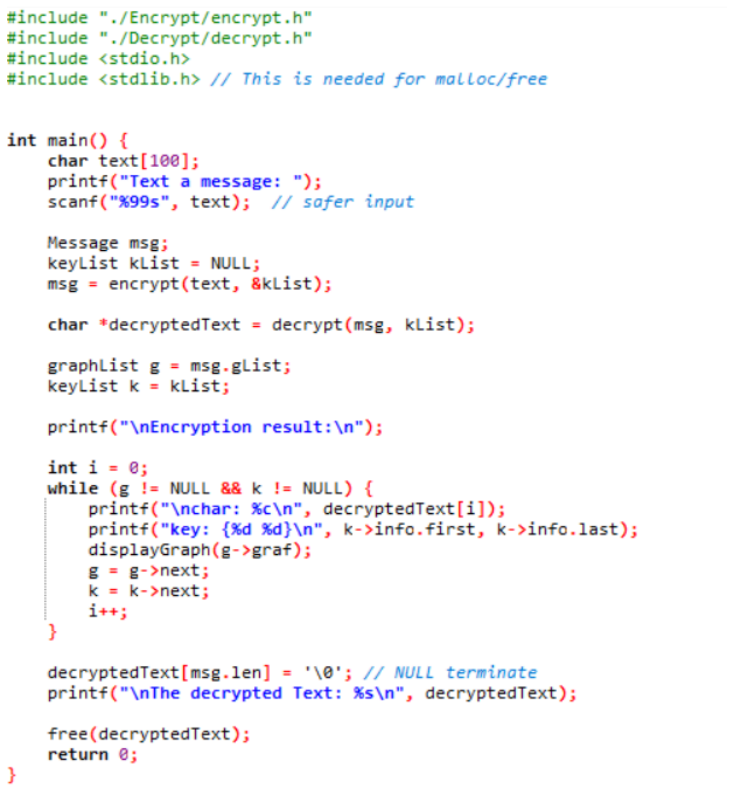
3. Start the program

```
./main
```

CODE:

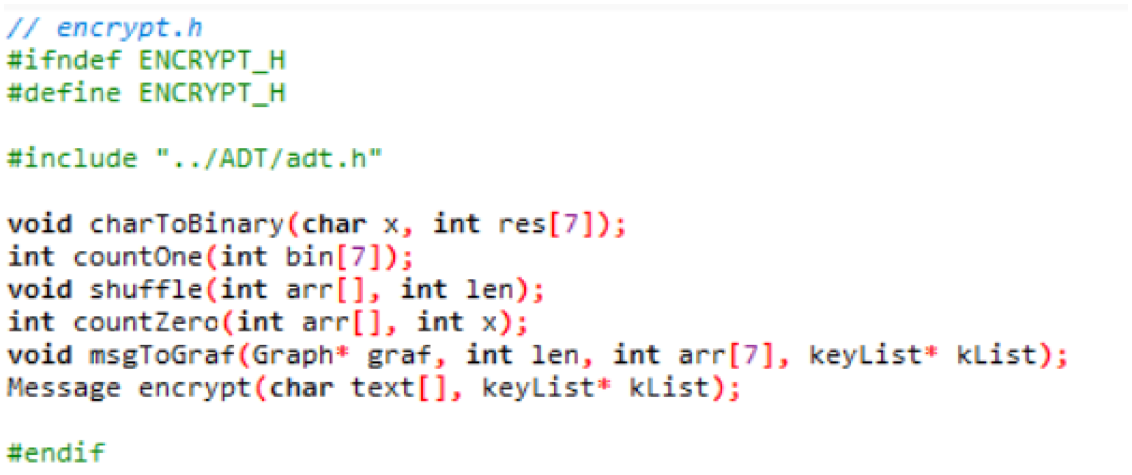
Input

main.c



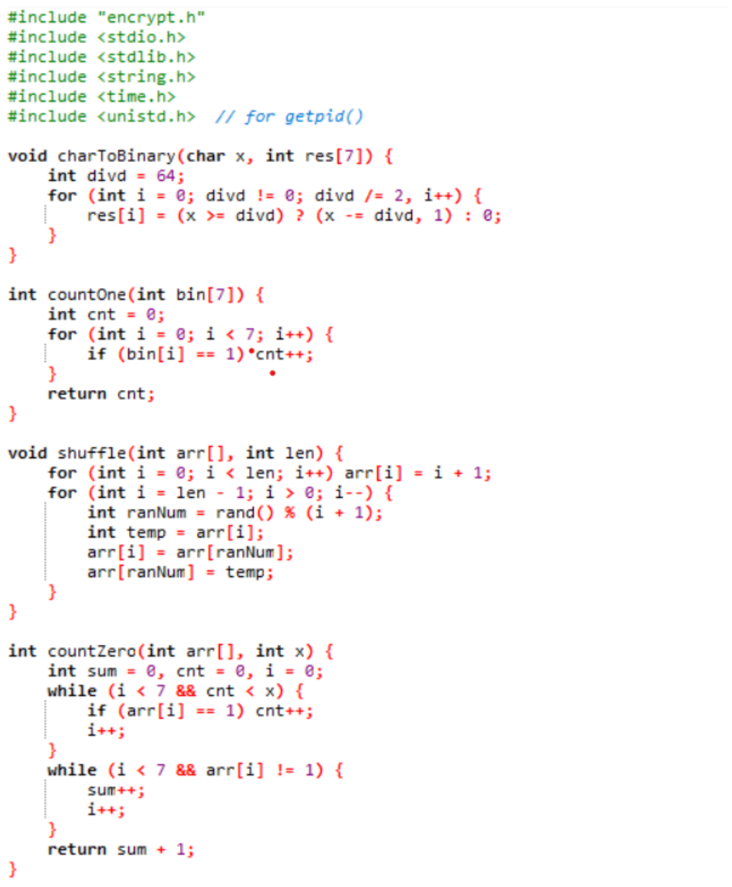
```c
#include "./Encrypt/encrypt.h"
#include "./Decrypt/decrypt.h"
#include <stdio.h>
#include <stdlib.h> // This is needed for malloc/free

int main() {
    char text[100];
    printf("Text a message: ");
    scanf("%99s", text);  // safer input

    Message msg;
    keyList kList = NULL;
    msg = encrypt(text, &kList);

    char *decryptedText = decrypt(msg, kList);

    graphList g = msg.gList;
    keyList k = kList;

    printf("\nEncryption result:\n");

    int i = 0;
    while (g != NULL && k != NULL) {
        printf("\nchar: %c\n", decryptedText[i]);
        printf("key: {%d %d}\n", k->info.first, k->info.last);
        displayGraph(g->graf);
        g = g->next;
        k = k->next;
        i++;
    }

    decryptedText[msg.len] = '\0'; // NULL terminate
    printf("\nThe decrypted Text: %s\n", decryptedText);

    free(decryptedText);
    return 0;
}
```

encrypt.h

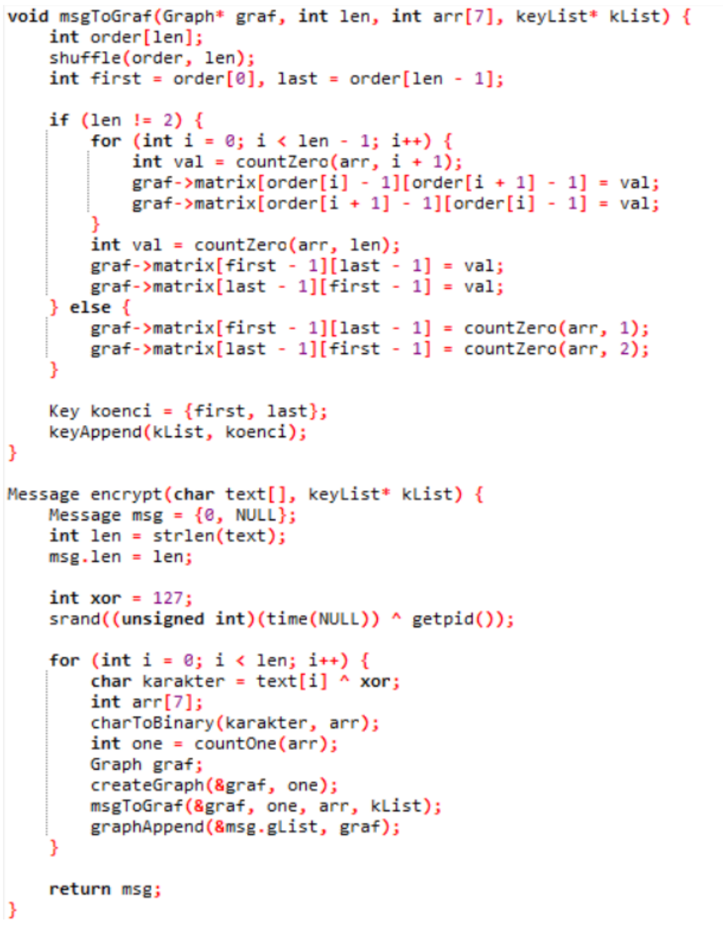


```c
// encrypt.h
#ifndef ENCRYPT_H
#define ENCRYPT_H

#include "../ADT/adt.h"

void charToBinary(char x, int res[7]);
int countOne(int bin[7]);
void shuffle(int arr[], int len);
int countZero(int arr[], int x);
void msgToGraf(Graph* graf, int len, int arr[7], keyList* kList);
Message encrypt(char text[], keyList* kList);

#endif
```

encrypt.c

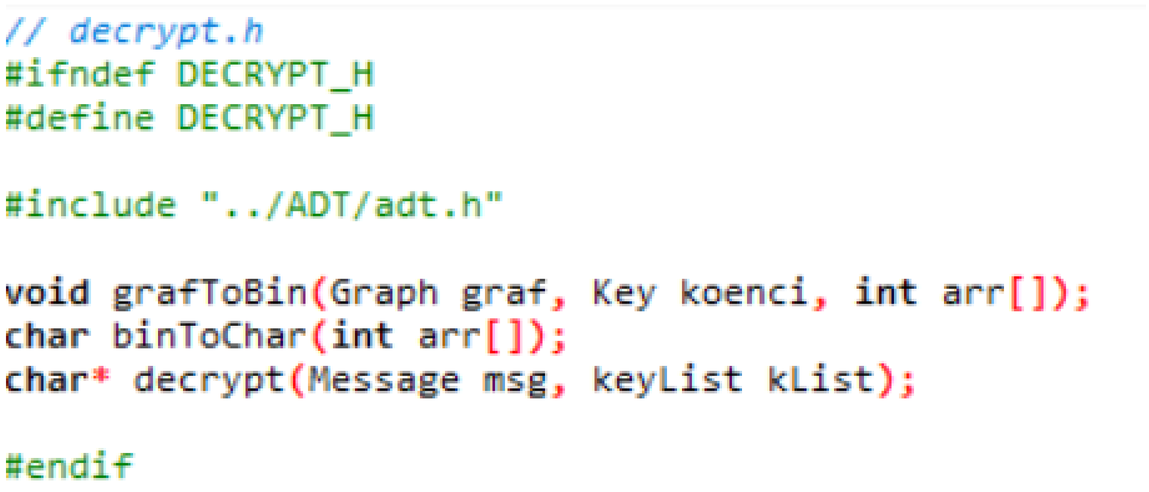


```c
#include "encrypt.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>  // for getpid()

void charToBinary(char x, int res[7]) {
    int divd = 64;
    for (int i = 0; divd != 0; divd /= 2, i++) {
        res[i] = (x >= divd) ? (x -= divd, 1) : 0;
    }
}

int countOne(int bin[7]) {
    int cnt = 0;
    for (int i = 0; i < 7; i++) {
        if (bin[i] == 1) cnt++;
    }
    return cnt;
}

void shuffle(int arr[], int len) {
    for (int i = 0; i < len; i++) arr[i] = i + 1;
    for (int i = len - 1; i > 0; i--) {
        int ranNum = rand() % (i + 1);
        int temp = arr[i];
        arr[i] = arr[ranNum];
        arr[ranNum] = temp;
    }
}

int countZero(int arr[], int x) {
    int sum = 0, cnt = 0, i = 0;
    while (i < 7 && cnt < x) {
        if (arr[i] == 1) cnt++;
        i++;
    }
    while (i < 7 && arr[i] != 1) {
        sum++;
        i++;
    }
    return sum + 1;
}
```



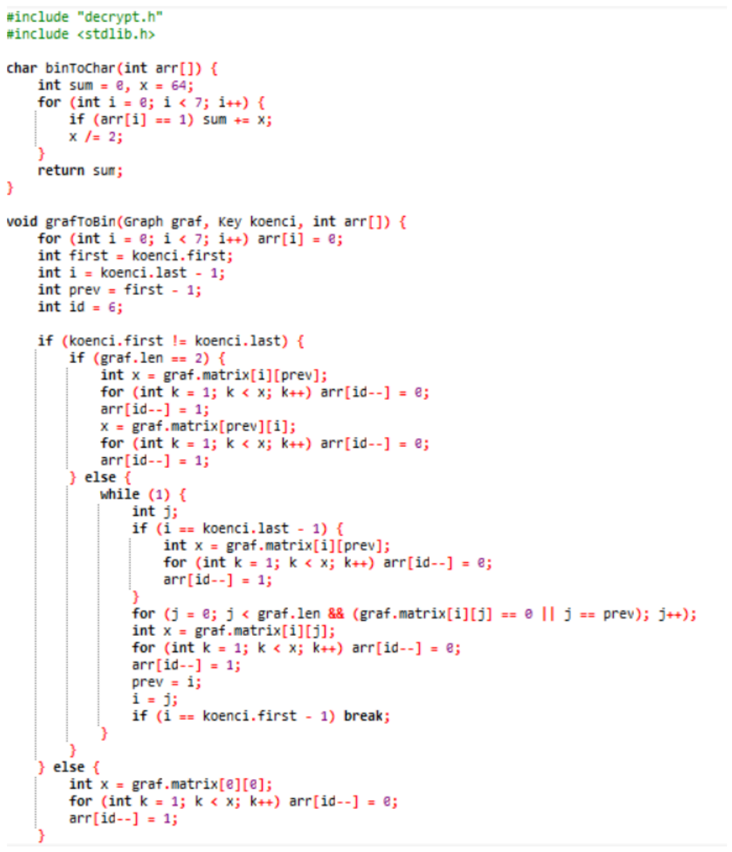
```c
void msgToGraf(Graph* graf, int len, int arr[7], keyList* kList) {
    int order[len];
    shuffle(order, len);
    int first = order[0], last = order[len - 1];

    if (len != 2) {
        for (int i = 0; i < len - 1; i++) {
            int val = countZero(arr, i + 1);
            graf->matrix[order[i] - 1][order[i + 1] - 1] = val;
            graf->matrix[order[i + 1] - 1][order[i] - 1] = val;
        }
        int val = countZero(arr, len);
        graf->matrix[first - 1][last - 1] = val;
        graf->matrix[last - 1][first - 1] = val;
    } else {
        graf->matrix[first - 1][last - 1] = countZero(arr, 1);
        graf->matrix[last - 1][first - 1] = countZero(arr, 2);
    }

    Key koenci = {first, last};
    keyAppend(kList, koenci);
}

Message encrypt(char text[], keyList* kList) {
    Message msg = {0, NULL};
    int len = strlen(text);
    msg.len = len;

    int xor = 127;
    srand((unsigned int)(time(NULL)) ^ getpid());

    for (int i = 0; i < len; i++) {
        char karakter = text[i] ^ xor;
        int arr[7];
        charToBinary(karakter, arr);
        int one = countOne(arr);
        Graph graf;
        createGraph(&graf, one);
        msgToGraf(&graf, one, arr, kList);
        graphAppend(&msg.gList, graf);
    }

    return msg;
}
```

<u>decrypt.h</u>

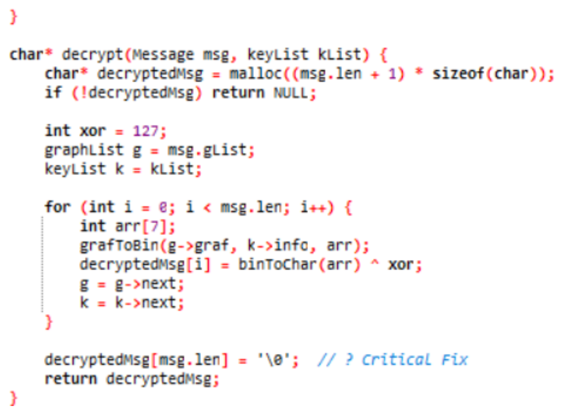


```c
// decrypt.h
#ifndef DECRYPT_H
#define DECRYPT_H

#include "../ADT/adt.h"

void grafToBin(Graph graf, Key koenci, int arr[]);
char binToChar(int arr[]);
char* decrypt(Message msg, keyList kList);

#endif
```

decrypt.c

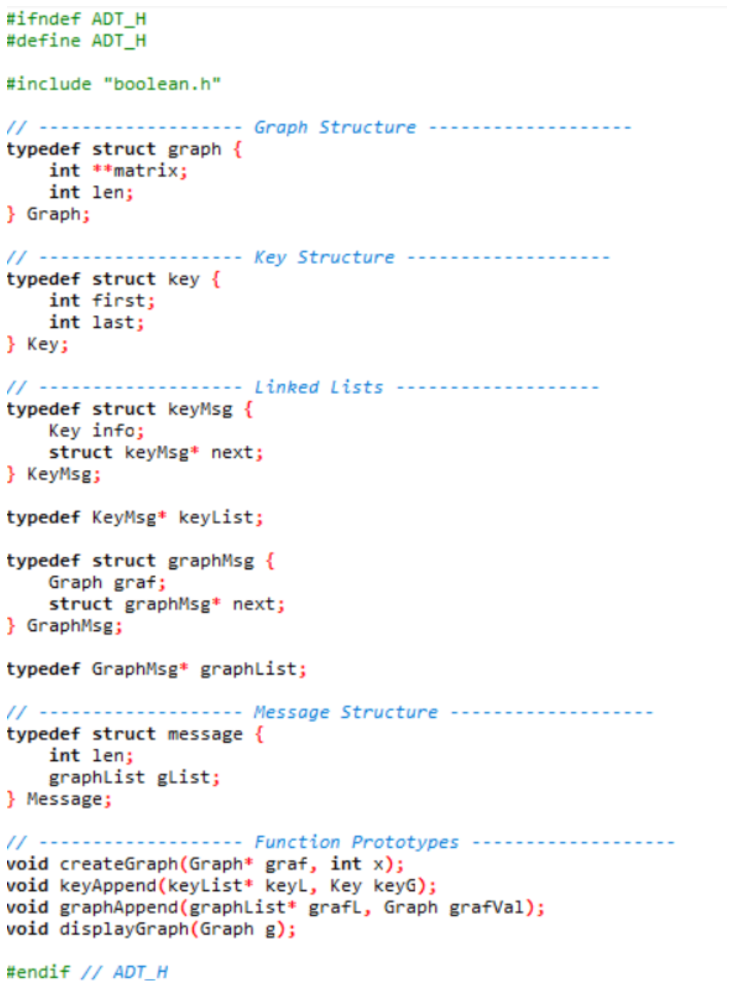


```c
#include "decrypt.h"
#include <stdlib.h>

char binToChar(int arr[]) {
    int sum = 0, x = 64;
    for (int i = 0; i < 7; i++) {
        if (arr[i] == 1) sum += x;
        x /= 2;
    }
    return sum;
}

void grafToBin(Graph graf, Key koenci, int arr[]) {
    for (int i = 0; i < 7; i++) arr[i] = 0;
    int first = koenci.first;
    int i = koenci.last - 1;
    int prev = first - 1;
    int id = 6;

    if (koenci.first != koenci.last) {
        if (graf.len == 2) {
            int x = graf.matrix[i][prev];
            for (int k = 1; k < x; k++) arr[id--] = 0;
            arr[id--] = 1;
            x = graf.matrix[prev][i];
            for (int k = 1; k < x; k++) arr[id--] = 0;
            arr[id--] = 1;
        } else {
            while (1) {
                int j;
                if (i == koenci.last - 1) {
                    int x = graf.matrix[i][prev];
                    for (int k = 1; k < x; k++) arr[id--] = 0;
                    arr[id--] = 1;
                }
                for (j = 0; j < graf.len && (graf.matrix[i][j] == 0 || j == prev); j++);
                int x = graf.matrix[i][j];
                for (int k = 1; k < x; k++) arr[id--] = 0;
                arr[id--] = 1;
                prev = i;
                i = j;
                if (i == koenci.first - 1) break;
            }
        }
    } else {
        int x = graf.matrix[0][0];
        for (int k = 1; k < x; k++) arr[id--] = 0;
        arr[id--] = 1;
    }
```



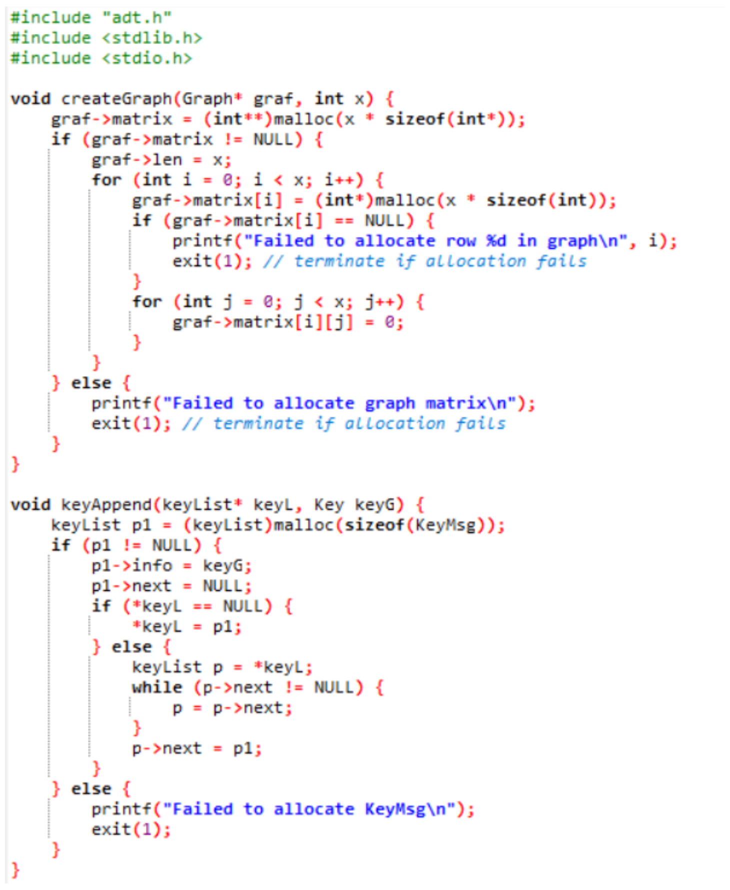
```c
}

char* decrypt(Message msg, keyList kList) {
    char* decryptedMsg = malloc((msg.len + 1) * sizeof(char));
    if (!decryptedMsg) return NULL;

    int xor = 127;
    graphList g = msg.gList;
    keyList k = kList;

    for (int i = 0; i < msg.len; i++) {
        int arr[7];
        grafToBin(g->graf, k->info, arr);
        decryptedMsg[i] = binToChar(arr) ^ xor;
        g = g->next;
        k = k->next;
    }

    decryptedMsg[msg.len] = '\0';  // ? Critical Fix
    return decryptedMsg;
}
```

adt.h

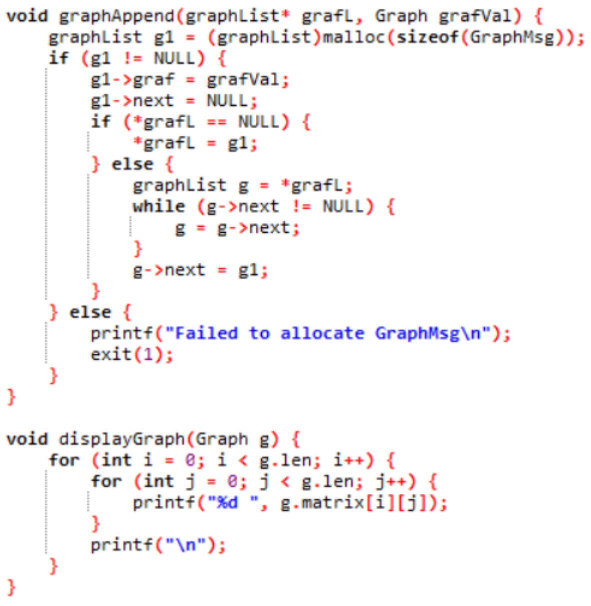


```c
#ifndef ADT_H
#define ADT_H

#include "boolean.h"

// ------------------- Graph Structure -------------------
typedef struct graph {
    int **matrix;
    int len;
} Graph;

// ------------------- Key Structure -------------------
typedef struct key {
    int first;
    int last;
} Key;

// ------------------- Linked Lists -------------------
typedef struct keyMsg {
    Key info;
    struct keyMsg* next;
} KeyMsg;

typedef KeyMsg* keyList;

typedef struct graphMsg {
    Graph graf;
    struct graphMsg* next;
} GraphMsg;

typedef GraphMsg* graphList;

// ------------------- Message Structure -------------------
typedef struct message {
    int len;
    graphList gList;
} Message;

// ------------------- Function Prototypes -------------------
void createGraph(Graph* graf, int x);
void keyAppend(keyList* keyL, Key keyG);
void graphAppend(graphList* grafL, Graph grafVal);
void displayGraph(Graph g);

#endif // ADT_H
```

adt.c

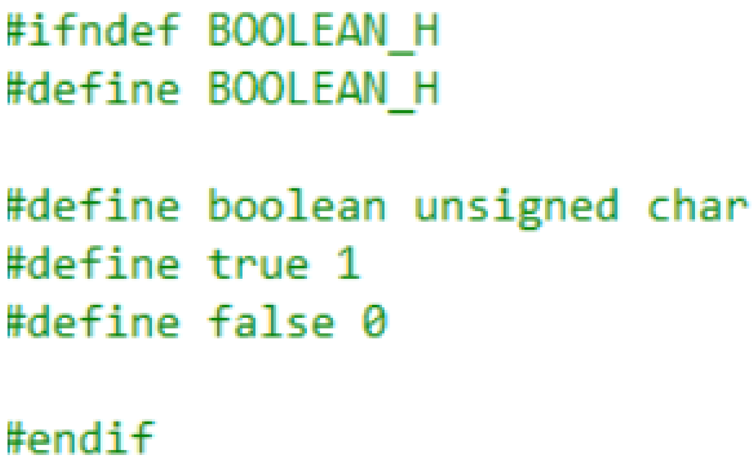


```c
#include "adt.h"
#include <stdlib.h>
#include <stdio.h>

void createGraph(Graph* graf, int x) {
    graf->matrix = (int**)malloc(x * sizeof(int*));
    if (graf->matrix != NULL) {
        graf->len = x;
        for (int i = 0; i < x; i++) {
            graf->matrix[i] = (int*)malloc(x * sizeof(int));
            if (graf->matrix[i] == NULL) {
                printf("Failed to allocate row %d in graph\n", i);
                exit(1); // terminate if allocation fails
            }
            for (int j = 0; j < x; j++) {
                graf->matrix[i][j] = 0;
            }
        }
    } else {
        printf("Failed to allocate graph matrix\n");
        exit(1); // terminate if allocation fails
    }
}

void keyAppend(keyList* keyL, Key keyG) {
    keyList p1 = (keyList)malloc(sizeof(KeyMsg));
    if (p1 != NULL) {
        p1->info = keyG;
        p1->next = NULL;
        if (*keyL == NULL) {
            *keyL = p1;
        } else {
            keyList p = *keyL;
            while (p->next != NULL) {
                p = p->next;
            }
            p->next = p1;
        }
    } else {
        printf("Failed to allocate KeyMsg\n");
        exit(1);
    }
}
```



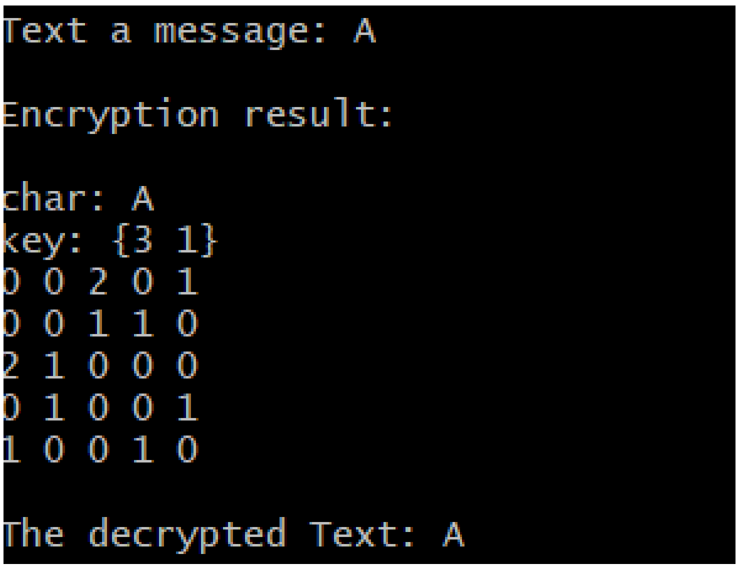
```c
void graphAppend(graphList* grafL, Graph grafVal) {
    graphList g1 = (graphList)malloc(sizeof(GraphMsg));
    if (g1 != NULL) {
        g1->graf = grafVal;
        g1->next = NULL;
        if (*grafL == NULL) {
            *grafL = g1;
        } else {
            graphList g = *grafL;
            while (g->next != NULL) {
                g = g->next;
            }
            g->next = g1;
        }
    } else {
        printf("Failed to allocate GraphMsg\n");
        exit(1);
    }
}

void displayGraph(Graph g) {
    for (int i = 0; i < g.len; i++) {
        for (int j = 0; j < g.len; j++) {
            printf("%d ", g.matrix[i][j]);
        }
        printf("\n");
    }
}
```

boolean.h



```c
#ifndef BOOLEAN_H
#define BOOLEAN_H

#define boolean unsigned char
#define true 1
#define false 0

#endif
```

Output



```
Text a message: A

Encryption result:

char: A
key: {3 1}
0 0 2 0 1
0 0 1 1 0
2 1 0 0 0
0 1 0 0 1
1 0 0 1 0

The decrypted Text: A
```

Results

- Successfully encrypted input text into a structured graph format.
- The decryption process accurately restored the original message.
- Demonstrated how graph-based data representation can enhance cryptographic security.

Conclusion

This project proves the applicability of Graph Theory in building secure cryptographic systems. By using graphs to encode and decode messages, we introduce an extra layer of complexity, making unauthorized decoding significantly harder. This technique is a foundational step towards building even more robust and intelligent security mechanisms in future systems.

Future Scope

- Implement weighted or labeled graphs for added encryption complexity.
- Introduce random graph generation for dynamic encoding.
- Integrate with file encryption systems.
- Apply in network security, especially in routing and secure messaging.