**File Encryption Tool - Complete Developer Documentation**

1. Algorithm / Workflow:

- Step 1: User chooses an action: Set Key, Encrypt, or Decrypt.

- Step 2: If "Set Key":

- Program asks for a secret key from the user.

- Key is stored in the global variable 'global Key' for encryption and decryption.

- Step 3: If "Encrypt":

- User provides a filename to encrypt.

- Program reads file contents byte by byte.

- Each byte is XORed with the corresponding byte of 'global Key'.

- Encrypted file is saved as <filename>.enc.

- Encryption key is automatically saved in <filename>.key.

- Step 4: If "Decrypt":

- User provides the encrypted file (\*.enc).

- Program extracts the original filename and reads the key from <original filename>.key.

- Each byte of the encrypted file is XORed with the key.

- Original content is restored in <original filename>.dec.

- If the key file is missing or corrupted, decryption fails.

- Step 5: Program ends or returns to the main menu.

2. Logic Used:

- Header Guards prevent multiple inclusion of header files.

- Global Variable 'global Key' is accessible across all modules.

- Modular Functions:

- setKey(): Sets or updates the key.

- encryptFile(): Encrypts the file and saves the key automatically.

- decryptFile(): Decrypts the file using the saved key.

- File Handling ensures secure reading and writing.

- XOR encryption requires the same key for decryption.

- Input validation is in place for filenames and keys.

3. Real-Time Problems:

- Wrong or missing key results in decryption failure.

- If the key file is deleted, decryption is not possible.

- An empty key makes encryption or decryption invalid.

- File permission errors show an error message.

- Large files may take longer due to byte-by-byte processing.

4. Technology Used:

- Language: C (Structured Programming)

- Header Files (.h) for function declarations and global variables

- Source Files (.c) for modular code

- Makefile (optional) for compiling with a single command

- Compiler: GCC (MSYS2, MinGW, WSL, Linux)

- GitHub for version control

- Cross-platform: Windows, Linux, macOS

- Modular design improves readability

- XOR-based encryption is lightweight and fast

- File I/O handling (rb, wb modes for binary safety)

- Terminal or command-line based UI (menu-driven)

5. New Developer Section:

5.1 Header Files:

- encryption.h

\* #ifndef ENCRYPTION\_H / #define ENCRYPTION\_H / #endif → Header guard to prevent multiple inclusion.

\* extern char global Key[200]; → Declares the global key variable defined in set-key.c

\* void encryptFile(const char \*filename); → Declaration of encryption function

\* void decryptFile(const char \*filename); → Declaration of decryption function

- decryption.h

\* #ifndef DECRYPTION\_H / #define DECRYPTION\_H / #endif → Header guard

\* void decryptFile(const char \*filename); → Declaration of decryption function

- set-key.h

\* #ifndef SET\_KEY\_H / #define SET\_KEY\_H / #endif → Header guard

\* extern char global Key[200]; → Same global key shared

\* void setKey(); → Declaration of function to set or update the key

5.2 Source Files:

main.c - Line-by-Line Explanation:

1 #include <stdio.h> → Required for printf(), scanf(), fopen(), fclose(), perror()

2 #include <stdlib.h> → Required for exit() function

3 #include <string.h> → Required for strlen(), strcpy(), strcmp(), etc.

4 #include "encryption.h" → Access encryptFile() and globalKey

5 #include "decryption.h" → Access decryptFile()

6 #include "set-key.h" → Access setKey() and globalKey

7 int main() { → Entry point of program

8 int choice; → Variable to store menu choice

9 char filename[100], key[100]; → Arrays for filename and key input

10 while(1) { → Infinite loop to keep showing the menu until exit

11 printf(...) → Display menu options

12 scanf("%d", &choice); → Take user input

13 switch(choice) { → Handle menu options

14 case 1: → Encrypt

15 printf("Enter filename..."); scanf("%s", filename);

16 encryptFile(filename); → Call encrypt function

17 break;

18 case 2: → Decrypt

19 printf("Enter encrypted filename..."); scanf("%s", filename);

20 decryptFile(filename); → Call decrypt function

21 break;

22 case 3: setKey(); break; → Call function to set the key

23 case 4: exit(0); → Exit program

24 default: printf("Invalid choice..."); break;

25 }

26 }

27 return 0;

28 }

encryption.c - Line-by-Line Explanation:

1 #include <stdio.h> → printf, fopen, fclose, fgetc, fputc, perror

2 #include <string.h> → strlen()

3 #include "encryption.h" → Function declaration

4 void encryptFile(const char \*filename) { → Function definition

5 FILE \*file = fopen(filename, "rb"); → Open file in binary read mode

6 if (!file) { perror("Error"); return; } → Check if file opened

7 char outputFilename[100]; → Output filename array

8 snprintf(outputFilename, sizeof(outputFilename), "%s.enc", filename); → Create encrypted filename

9 FILE \*outputFile = fopen(outputFilename, "wb"); → Open output file

10 if (!outputFile) { perror("Error"); fclose(file); return; }

11 int keyLength = strlen(globalKey); → Get length of key

12 int ch, i = 0; → Loop variables

13 while ((ch=fgetc(file)) != EOF) { fputc(ch ^ globalKey[i % keyLength], outputFile); i++; } → XOR encryption

14 fclose(file); fclose(outputFile);

15 FILE \*keyFile = fopen(outputFilename + ".key", "w"); → Save key in .key file

16 fprintf(keyFile, "%s", globalKey); fclose(keyFile);

17 printf("File encrypted successfully!");

18 }

decryption.c - Line-by-Line Explanation:

1 #include <stdio.h> → Standard I/O

2 #include <string.h> → String functions

3 #include "decryption.h"

4 void getOriginalFilename(...) → Removes .enc extension

5 void decryptFile(const char \*filename) {

6 char originalFilename[100]; getOriginalFilename(filename, originalFilename);

7 char keyFilename[100]; snprintf(keyFilename, sizeof(keyFilename), "%s.key", originalFilename);

8 FILE \*keyFile = fopen(keyFilename, "r"); → Read key

9 fscanf(keyFile, "%s", globalKey); fclose(keyFile);

10 FILE \*file = fopen(filename, "rb"); → Open encrypted file

11 char outputFilename[100]; snprintf(outputFilename, sizeof(outputFilename), "%s.dec", originalFilename);

12 FILE \*outputFile = fopen(outputFilename, "wb");

13 int keyLength = strlen(globalKey), ch, i=0;

14 while ((ch=fgetc(file))!=EOF) { fputc(ch ^ globalKey[i % keyLength], outputFile); i++; }

15 fclose(file); fclose(outputFile);

16 printf("File decrypted successfully! Output: %s\n", outputFilename);

17 }

set-key.c - Line-by-Line Explanation:

1 #include <stdio.h>

2 #include <string.h>

3 #include "set-key.h"

4 void setKey() {

5 printf("Enter a new key: "); scanf("%s", globalKey); → Store input in globalKey

6 printf("Key set successfully!");

7 }

6. Terminal / Command Line Usage:

- Compile Linux or macOS:

gcc main.c encryption.c decryption.c set-key.c -o FileEncryptionTool

./FileEncryptionTool

- Compile Windows (MSYS2, MinGW, WSL):

gcc main.c encryption.c decryption.c set-key.c -o FileEncryptionTool.exe

.\FileEncryptionTool.exe

Explanation:

\* gcc → GNU Compiler

\* -o → Specify output filename

\* ./ or .\ → Run executable from the current folder

\* exit() → Terminate program

\* scanf("%s", variable) → Read input

\* printf() → Display output

\* fopen(filename, mode) → Open file

\* fgetc and fputc → Read and write a single byte

\* fclose → Close file

7. Summary:

- The key is automatically saved for consistent encryption and decryption.

- The global variable 'global Key' is used across files.

- Modular code separates encryption, decryption, and key management.

- The system handles real-time file and key errors.

- It's easy for new developers to grasp the workflow.

- It uses lightweight XOR encryption for speed and simplicity.

- The tool supports multiple platforms.

- It features a terminal-based menu interface.