PROJECT : IMCRYPT – IMAGE ENCRYPTOR

(_Explainer file_)

Index.html ⟶ Gives basic structure to webpage

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Imcrypt</title>
    <link rel="stylesheet" href="style.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link
href="https://fonts.googleapis.com/css2?family=Kode+Mono:wght@400..700&display=swap"
rel="stylesheet">

    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Kaushan+Script&display=swap"
rel="stylesheet">
</head>
<body>
    <h1>Imcrypt</h1>
    <div class = container>
        <button id="encryptButton">Encrypt</button>
        <button id="decryptButton">Decrypt</button>
    </div>

    <script>
        const encryptButton = document.getElementById('encryptButton');
        const decryptButton = document.getElementById('decryptButton');

        encryptButton.addEventListener('click', async () => {
            const filePath = await window.electron.openFileDialog();
            const outputImageFileName = await
window.electron.saveFileDialog('encrypted.png');
            const outputKeyFileName = await window.electron.saveFileDialog('key.txt');
            await window.electron.encrypt(filePath, outputImageFileName, outputKeyFileName);
            alert('Encryption Complete');
        });

        decryptButton.addEventListener('click', async () => {
            const filePath = await window.electron.openFileDialog();
            const keyPath = await window.electron.openFileDialog();
            const outputImageFileName = await
window.electron.saveFileDialog('decrypted.png');
            await window.electron.decrypt(filePath, keyPath, outputImageFileName);
```

Google fonts

Buttons

Adding click events to buttons.
- Open file
- encrypt
- save

→ Using dialouge box

functions that encrypts and decrypts. stored in encrypt.js & decrypt.js

```
            alert('Decryption Complete');
        });
    </script>
</body>
</html>
```

Index.js → To initialise electron application and creates main application window.

```
const { app, BrowserWindow } = require('electron'); Importing electron module
const path = require('path'); importing path module
require('./gui'); // Require the GUI module to register IPC handlers

function createWindow() {
    const mainWindow = new BrowserWindow({
        width: 800,
        height: 600,        To create a new browser window using electron
        webPreferences: {
            preload: path.join(__dirname, 'preload.js')                    application
        }
    });                  ↑ Webprefrence let's setup setting before using ↵
                           Here we're loading all in preload.js

    mainWindow.loadFile(path.join(__dirname, 'index.html')); loads index.html
}                                                            structure in that
                                                             window.
app.on('ready', createWindow); Creates window when app is ready

app.on('window-all-closed', () => {
    if (process.platform !== 'darwin') {
        app.quit();
    }                    Quits app when windows
});                          are closed.

app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
        createWindow();          Window
    }                  Creates   ^  when app is
});                    active and no window is open.
```

Preload.js → Bridge the communication between webpage and electron main process.

```
const { contextBridge, ipcRenderer } = require('electron');

contextBridge.exposeInMainWorld('electron', {
    encrypt: (filePath, outputImageFileName, outputKeyFileName) =>
ipcRenderer.invoke('encrypt', filePath, outputImageFileName, outputKeyFileName),
    decrypt: (filePath, keyPath, outputImageFileName) => ipcRenderer.invoke('decrypt',
filePath, keyPath, outputImageFileName),
    openFileDialog: () => ipcRenderer.invoke('open-file-dialog'),
    saveFileDialog: (defaultPath) => ipcRenderer.invoke('save-file-dialog', defaultPath)
});
```

Importing necessary electron library.

Defining functions

gui.js → Handles inter-process communication for encrypting / decrypting.

```
const { ipcMain, dialog } = require('electron');
const path = require('path');
const encrypt = require('./encrypt');
const decrypt = require('./decrypt');

ipcMain.handle('encrypt', async (event, filePath, outputImageFileName, outputKeyFileName) =>
{
    await encrypt({ e: filePath, o: outputImageFileName, p: outputKeyFileName });
});

ipcMain.handle('decrypt', async (event, filePath, keyPath, outputImageFileName) => {
    await decrypt({ d: filePath, k: keyPath, o: outputImageFileName });
});

ipcMain.handle('open-file-dialog', async () => {
    const result = await dialog.showOpenDialog({ properties: ['openFile'] });
    return result.filePaths[0];
});

ipcMain.handle('save-file-dialog', async (event, defaultPath) => {
    const result = await dialog.showSaveDialog({ defaultPath });
    return result.filePath;
});
```

Imports

Calls encrypt function with provided paths

Calls decrypt function with provided paths

opens 'open-file' dialogue box.

opens 'save-file' dialogue box.

Encrypt.js → file that encrypts

```
const alert = require('cli-alerts'); — for alerts
const fs = require('fs'); — for file processing
const jimp = require('jimp'); — for image processing
const path = require('path'); — for path processing

                              ← argument → path of file
const encrypt = async (flags) => {
    const filePath = flags.e;
    if (!filePath) {
        alert({
            type: 'warning',
            name: 'Invalid file path',
            msg: 'Please provide a valid file path'
        });
        throw new Error('Invalid file path');
    }

    const fullPath = path.resolve(filePath);
    if (!fs.existsSync(fullPath)) {
        alert({
            type: 'warning',
            name: 'Invalid file path',
            msg: 'Please provide a valid file path'
        });
        throw new Error('Invalid file path');
    }

    try {
        const image = await jimp.read(fullPath);
        const extension = image.getExtension();
        const fileNameWithoutExtension = path.basename(fullPath, path.extname(fullPath));

        let outputImageFile = `${fileNameWithoutExtension}_encrypted.${extension}`;
        if (flags.o) {
            outputImageFile = path.resolve(flags.o);
        }

        if (fs.existsSync(outputImageFile)) {
            alert({
                type: 'error',
                name: 'Invalid output image file name',
                msg: `The output image file name already exists: ${outputImageFile}`
            });
            throw new Error('Invalid output image file name');
        }

        let outputKeyFile = `${fileNameWithoutExtension}_key.txt`;
```

Imports

→ Extract the path from flag and check if that is provided or not

→ Resolves the file path. (absolute ↔ relative)

→ Check if file exists at that location or not.

— Reads image using jimp

— Creates output file name for encrypted image & key.

— Throws errors if encounters anything else.

```
        if (flags.p) {
            outputKeyFile = path.resolve(flags.p);
        }

        if (fs.existsSync(outputKeyFile)) {
            alert({
                type: 'error',
                name: 'Invalid output key file name',
                msg: `The output key file name already exists: ${outputKeyFile}`
            });
            throw new Error('Invalid output key file name');
        }

        const rgba = image.bitmap.data;
        const length = rgba.length;
        const key = [];
        for (let i = 0; i < length; i++) {
            key.push(Math.floor(Math.random() * 256));
        }

        for (let i = 0; i < length; i++) {
            const k = key[i];
            rgba[i] = rgba[i] ^ k;
        }

        image.bitmap.data = rgba;
        image.write(outputImageFile);

        fs.writeFileSync(outputKeyFile, Buffer.from(key).toString('base64'));

        alert({
            type: 'success',
            name: 'Image encrypted successfully',
            msg: `Image encrypted successfully:\nEncrypted Image: ${outputImageFile}\nKey:
${outputKeyFile}`
        });
    } catch (error) {
        alert({
            type: 'error',
            name: 'Error',
            msg: error.message || 'Unknown error'
        });
        throw error;
    }
};

module.exports = encrypt;
```

- extract RGBA data from image
- gets its length
- generate random key of same length.

- encrypt image by xoring each byte with corresponding byte from key.
- Writes the encrypted img.

← writes encryption key

↑ prompt success

← else error thrown

↳ export encrypt function so that it can be used in other file.

Decrypt.js → file that decrypts

```
const alert = require('cli-alerts');
const fs = require('fs');
const jimp = require('jimp');
const path = require('path');

const decrypt = async (flags) => {
    const filePath = flags.d;
    const keyPath = flags.k;
    if (!filePath || !keyPath) {
        alert({
            type: 'warning',
            name: 'Invalid file path',
            msg: 'Please provide a valid file path and key file path'
        });
        throw new Error('Invalid file path or key file path');
    }

    const fullPath = path.resolve(filePath);
    const fullKeyPath = path.resolve(keyPath);

    if (!fs.existsSync(fullPath) || !fs.existsSync(fullKeyPath)) {
        alert({
            type: 'warning',
            name: 'Invalid file path',
            msg: 'Please provide a valid file path and key file path'
        });
        throw new Error('Invalid file path or key file path');
    }

    try {
        const image = await jimp.read(fullPath);
        const keyBase64 = fs.readFileSync(fullKeyPath, 'utf8');
        const key = Buffer.from(keyBase64, 'base64');

        if (key.length !== image.bitmap.data.length) {
            alert({
                type: 'error',
                name: 'Invalid key',
                msg: 'The key length does not match the image data length'
            });
            throw new Error('Invalid key length');
        }

        const rgba = image.bitmap.data;
        const length = rgba.length;
```

importing modules

Takes as argument from that extract and check if Image & key path are present or not if not throw error.

Resolve paths and check if file exists or not at passed path.

— Read image & keys

— check if keys length matches, image data length for Xoring.

→ read image data

→ figure out its length

```
        for (let i = 0; i < length; i++) {
            const k = key[i];
            rgba[i] = rgba[i] ^ k;
        }

        image.bitmap.data = rgba;

        let outputImageFile = path.resolve(filePath).replace('_encrypted', '_decrypted');
        if (flags.o) {
            outputImageFile = path.resolve(flags.o);
        }

        if (fs.existsSync(outputImageFile)) {
            alert({
                type: 'error',
                name: 'Invalid output image file name',
                msg: `The output image file name already exists: ${outputImageFile}`
            });
            throw new Error('Invalid output image file name');
        }

        image.write(outputImageFile);

        alert({
            type: 'success',
            name: 'Image decrypted successfully',
            msg: `Image decrypted successfully:\nDecrypted Image: ${outputImageFile}`
        });
    } catch (error) {
        alert({
            type: 'error',
            name: 'Error',
            msg: error.message || 'Unknown error'
        });
        throw error;
    }
};

module.exports = decrypt;
```

xor it with key to decrypt.

Replace image data with new xored data.

Generate formatted o/p file name

while saving check if filename already exists.

if not writes/saves

prompt "success"

else thrown error if any for eg: user cancelled it.

export decrypt function so that it can be used in other files.

node_modules Folder

- This folder contains all the dependencies and modules required for the project. When you install a package using npm, it gets stored in this folder.

package.json

- This file holds metadata relevant to the project and lists the dependencies needed. It is essential for managing the project and ensuring that everyone working on the project uses the same dependencies.

package-lock.json

- This file is automatically generated when npm modifies either node_modules or package.json. It ensures that the exact version of dependencies is installed in case the project is shared or deployed, providing consistency across different environments.
