A PROJECT REPORT

On

CIPHER SHIELD

Submitted in partial fulfillment of the requirement of University of Mumbai for the Degree of

Bachelor of Engineering

In

Computer Science and Engineering

Submitted By

Aayush Jadhav

Sameet Jathan

Soham Kadam

Aditya Kadav

Supervisor

Supervisor Name

Department Of Computer Science and Engineering

XAVIER INSTITUTE OF ENGINEERING

Mahim – 400016

UNIVERSITY OF MUMBAI

Academic Year 2023 – 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
XAVIER INSTITUTE OF ENGINEERING
Mahim– 400016

CERTIFICATE

This is to certify that the requirements for the project report entitled '**CIPHER SHIELD**' have been successfully completed by the following students:

Name	**Roll No.**
Aayush Jadhav	20
Sameet Jathan	21
Soham Kadam	22
Aditya Kadav	23

in partial fulfillment of Bachelor of Engineering of Mumbai University in the Department of Computer Science and Engineering, Xavier Institute of Engineering, Mahim-400016 during the Academic Year 2023 – 2024.

Supervisor
(Name of Supervisor)

Head, Department Computer Science and Engg.
(Dr.Ninad More)

Principal
(Dr. Y.D. Venkatesh)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

XAVIER INSTITUTE OF ENGINEERING

Mahim– 400016

PROJECT APPROVAL FOR S.E.

This project entitled "Cipher Shield" by Aayush Jadhav, Sameet Jathan, Soham Kadam and Aditya Kadav are approved for the degree of Bachelor of Engineering in Computer Science and Engineering

Examiners:

1. ________________

2. ________________

Supervisors:

1. ________________

2. ________________

Chairman:

1. ________________

Date:

Place:

Table of Contents

1.	Introduction			1
2.	Project Implementation			5
	2.1	Overview		6
		2.1.1	Existing Systems	7
		2.1.2	Proposed System	8
	2.2	Implementation Details		
		2.2.1	Hardware and Software Details	9
3	Project Inputs and Outputs			
	3.1	Input Details Outputs/Screen Shots		14
4.	Conclusion and Future Scope			
	4.1	Conclusion		32
	4.2	Future Scope		33
References				34

1 Introduction

In the digital age, where sensitive information is constantly at risk from cyber threats, data security has become an absolute necessity. As we rely more and more on digital storage and transmission of data, it is imperative to safeguard our information from unauthorized access. Cipher Shield is a comprehensive solution that addresses this critical need by providing a robust and user-friendly system for file encryption and decryption.

In an era marked by the rapid exchange of data over the internet, the risk of data breaches and unauthorized access is ever-present. Hackers, malware, and even the inadvertent loss or theft of devices pose substantial threats to the confidentiality and integrity of our data. As such, Cipher Shield emerges as a powerful tool for both individuals and organizations to secure their digital assets.

Cipher Shield

The primary aim of this project report is to delve into the features, functionality, and significance of Cipher Shield in the realm of data security. Cipher Shield is designed to offer an intuitive and secure solution for encrypting files of various formats, rendering them inaccessible to anyone lacking the proper decryption keys. This protective measure ensures that only authorized users can access and make sense of the encrypted data, thus guaranteeing the privacy and security of sensitive information.

What is Encryption

It’s the process of transforming readable data into an unreadable format. It uses mathematical algorithms to achieve this and can be performed on any kind of data, including emails, documents, and messages. Encryption is often used in the digital world, where it’s applied to files, emails, and even entire hard drives. It’s also used in securing wireless networks and is the backbone of many online banking systems. Without encryption, our data would be vulnerable to anyone who wants to see it.

Another thing to consider is that encryption keys are generated by two different algorithms — symmetrical and asymmetrical. Symmetrical algorithm allows for a shared key between users on both ends, while asymmetrical algorithm requires two keys (one public and one private) for each user. This means that even if someone obtains your public key, they still need your private key to decrypt your data — adding an extra layer of security.

What is Decryption

Decryption is the process of transforming encrypted data into its original form so that humans can read it. This is done by reversing the encryption process, using the same algorithm, key, or password.

There are many different applications for decryption, including recovering lost or forgotten passwords, accessing encrypted data for analysis, and restoring files that have been damaged or corrupted. In some cases, decryption may also be used to remove security measures from files or systems so that they can be used without restriction.

Keypoint Differences between Encryption and Decryption

Encryption is the process of transforming normal data into an unreadable form and decryption is the process of transforming unreadable/encoded data back into its original form.

Encryption is done by the person sending the data to the destination, while decryption is done by the person receiving the data.

Encryption is ‘scrambling’ data, making it unreadable to those without a special ‘key’ to unlock it. On the other hand, decryption is simply reversing the encryption process and making the data readable again.

Whenever data is sent between two separate machines, it is automatically encrypted with a private key. In decryption the recipient of the data automatically allows the conversion of the data from code to its original form.

Examples of encryption are: An employee sends an encrypted document to his boss. An example decryption is: Boss decryptes the document sent by employer.

Why Encryption and Decryption are Important

Encryption and decryption are important because they keep your data safe. When you encrypt your data, it means that only the people you want to be able to see it will be able to. It’s like locking your data away in a safe where only you have the key.

Decryption is the process of unlocking your data so that it can be read. It’s like opening up the safe and taking out your data so that you can use it. Your data will be safe from prying eyes as long as encrypted.

Difference between Encryption and Decryption

Encryption	Decryption
Encryption is the process of converting normal message into meaningless message.	While decryption is the process of converting meaningless message into its original form.
Encryption is the process which take place at sender's end.	While decryption is the process which take place at receiver's end.
Its major task is to convert the plain text into cipher text.	While its main task is to convert the cipher text into plain text.
Any message can be encrypted with either secret key or public key.	Whereas the encrypted message can be decrypted with either secret key or private key.
In encryption process, sender sends the data to receiver after encrypted it.	Whereas in decryption process, receiver receives the information (Cipher text) and convert into plain text.
The same algorithm with the same key is used for the encryption-decryption process.	The only single algorithm is used for encryption-decryption with a pair of keys where each use for encryption and decryption.
Encryption is used to protect the confidentiality of data by converting it into an unreadable form that can only be read by authorized parties.	Decryption is used to reverse the encryption process and convert the ciphertext back into plaintext.
The output of encryption is a ciphertext that is unintelligible to anyone who does not have the decryption key.	The output of decryption is the original plaintext message.

2 Project Implementation

The Cipher Shield project aims to create a secure and user-friendly application for encrypting and decrypting files using various encryption algorithms. This project ensures data confidentiality and security, making it a valuable tool for safeguarding sensitive information.

Implementation :-

1. **Encryption Algorithms:** The JavaScript code used to encrypt the file is AES-GCM encryption algorithm. This code generates a random encryption key and an Initialization Vector (IV) for each file and then encrypts the file's contents. It also allows users to download the encrypted file and the encryption key.
2. **User Authentication and Authorization:** The algorithm used in Cipher Shield to encrypt the files provides secured interface to encrypt a file and only allows the authorized user to access the encrypted file by providing the authentication key .
3. **File Upload and Encryption Process:** The file can be easy uploaded by the user, using the ‘Upload File’ button and then the selected file will seamlessly get uploaded in the webpage, then the webpage will provide a ‘Encrypt file’ button, by clicking on it user can encrypt the file and then download the encrypted file and it’s key for safe use.
4. **Testing and Quality Assurance:** The Cipher Shield Website has been used and tested several times by the developers to provide seamless and easy interface to final users.

2.1 Overview

In an age where data security and privacy have become paramount, the need for robust file encryption and decryption solutions is more critical than ever. "Cipher Shield" represents a mini project that aims to address this pressing concern by offering a secure and user-friendly application for encrypting and decrypting files.

Project Objectives :-

1. **File Encryption:** Cipher Shield provides a seamless process to encrypt sensitive files. It employs advanced encryption algorithms to ensure the confidentiality and integrity of the data.
2. **File Decryption:** The project allows authorized users to decrypt the encrypted files, restoring them to their original state, ensuring accessibility while maintaining security.
3. **User Authentication:** Cipher Shield incorporates user authentication mechanisms to restrict access to authorized users only. This ensures that encrypted files remain secure.
4. **Intuitive User Interface:** A user-friendly graphical interface is developed to make the encryption and decryption process easy and accessible for users with various levels of technical expertise.

2.1.1 Existing Systems

There are several websites and online tools that allow you to encrypt and decrypt files. However, it's important to note that using online tools for encryption and decryption may not always be the most secure option, and there are potential drawbacks to consider.

Websites :- Online-Convert.com
:- FileEncryptor.com
:- AxCrypt Online
:- Mega.nz

Drawbacks of Using Online Encryption Services:

- **Security Concerns:** Uploading sensitive or confidential files to an online service poses security risks. If the service gets compromised, your data could be at risk.
- **Lack of Control:** When you use online services, you relinquish some control over the encryption process. You have to trust the service provider to handle your data securely.
- **Limited Features:** Online encryption services may not offer advanced features or customization options that you might find in standalone encryption software.
- **File Size Limits:** Many online services impose file size limits, which can be a problem if you need to encrypt large files.
- **Internet Connection:** You need a stable internet connection to use these services, which may not be practical in all situations.

For highly sensitive data or for situations where security is paramount, it's often better to use standalone encryption software or tools installed on your local device. These tools provide more control over the encryption process and reduce the risks associated with using online services.

2.1.2 Proposed Systems

Cipher shield provide one stop solution to convert all types of your file in one go whether it is plain text ,word file ,image file, video file, Excel sheet. Cipher Shield can Decrypt and Encrypt all types of your file in a seamless and easy way

General advantages and Pros of using "Cipher Shield" – File Encryption and Decryption website :-

1. **Security Features:** Look for strong encryption methods, such as AES (Advanced Encryption Standard), and ensure that the service uses robust encryption algorithms and key management practices. Your data's security should be a top priority.
2. **User-Friendly Interface:** A good encryption service should have an intuitive and user-friendly interface, making it easy for users to encrypt and decrypt files without technical expertise.
3. **Data Privacy:** Ensure the service has a clear privacy policy and does not access or store your data. Ideally, it should offer end-to-end encryption, meaning only you have access to the decryption key.
4. **Compatibility:** Check whether the service supports a variety of file formats and is compatible with the devices and platforms you use, such as Windows, macOS, iOS, Android, etc.
5. **File Size Limits:** Verify if the service imposes any file size limits that may affect your needs. Some services have restrictions on the size of files that can be encrypted.
6. **Password Protection:** The service should allow you to set strong and unique passwords for encrypting files. Strong password policies enhance security.

2.2 Implementation Details

2.2.1 Hardware Details

Server Infrastructure: Cipher Shield is hosted on a dedicated server to ensure data security and performance. The server hardware includes:

- Intel i5 Processor
- 8 GB – 16 GB RAM
- 512 GB SSD Storage
- 10 MB/s Network

Security Hardware: To enhance the security of the file encryption and decryption processes, Cipher Shield employs hardware security modules (HSMs). These HSMs provide cryptographic processing and key management.

Load Balancing: To distribute incoming traffic and maintain high availability, Cipher Shield uses load balancers.

Scalability and Future Expansion: Hardware used currently can be updated in the future. The websites can be made more user-friendly and can be made for mobile phones.

Client-Side Hardware Requirements: User can use any type of PC or Laptop, the Cipher Shield’s adaptive nature allows the website to run on any PC or Laptop needless of it’s configuration.

2.2.1 Software Details

The Frontend of the Cipher Shield website is made by using HTML programming language along with CSS programming language which is use to design the webpages of the Cipher Shield website.

HTML is used because it is the easiest language to create a seamless and user-friendly website. CSS is used coz it provides seamless codes to design the webpages created using HTML.

The Backend of the Cipher Shield website is compiled using Java and JavaScript along with AES-GCM encryption algorithm which is use to encrypt the user's files

Programming Languages like Java, JavaScript as it is easy to implement and it provides very simple AES-GCM encryption algorithm to encrypt files

AES-GCM, which stands for Advanced Encryption Standard in Galois/Counter Mode, is a widely used encryption algorithm that provides both confidentiality and integrity for data. AES-GCM is a symmetric encryption algorithm that provides both encryption and authentication (integrity checking) of the data.

AES Encryption (Advanced Encryption Standard): AES is a symmetric-key encryption algorithm that uses a secret key for both encryption and decryption. AES operates on fixed-size blocks of data (typically 128 bits or 16 bytes) and uses a specific key length (128, 192, or 256 bits).

GCM Mode (Galois/Counter Mode): GCM is a mode of operation for block ciphers like AES. It combines the encryption capabilities of AES with a counter mode of operation and additional data for authentication. GCM is designed to provide data confidentiality and data origin authentication (integrity) simultaneously.

To encrypt the PDF and Text Documents we used the Web Crypto API to perform PDF and Text Document encryption, and it employs the "AES-GCM" (Advanced Encryption Standard in Galois/Counter Mode) encryption algorithm. Specifically, it utilizes the AES-GCM algorithm for encrypting the PDF file and Text document.

Here's a breakdown of the key parts in the code that involve AES-GCM encryption:

1. **Importing the Encryption Key:** The code generates a random encryption key of 128 bits (16 bytes) using crypto.getRandomValues(encryptionKey) and then imports this key using the Web Crypto API. The imported key is used for encryption.
2. **Encryption:** The code uses the imported AES-GCM key to encrypt the contents of the selected text file. This is done with the crypto.subtle.encrypt function, which uses the AES-GCM algorithm for encryption.

To decrypt the PDF and Text Document we used the the "AES-GCM" (Advanced Encryption Standard in Galois/Counter Mode) decryption algorithm.

Here's a breakdown of the key parts in the code that involve AES-GCM decryption:

1. **Importing the Decryption Key:** The code prompts the user to enter the encryption key, which should be a 32-character hexadecimal string (128 bits or 16 bytes). It then converts this hexadecimal string to a Uint8Array to represent the decryption key. The key is imported as a CryptoKey for decryption.
2. **Decryption:** The code uses the imported AES-GCM key to decrypt the contents of the selected encrypted file. This is done with the crypto.subtle.decrypt function, which uses the AES-GCM algorithm for decryption.

AES-GCM Encryption Process:

Input: The encryption process starts with the plaintext (the data you want to encrypt), a secret encryption key (a random sequence of bits), and an Initialization Vector (IV) as input.
Key Expansion: The secret encryption key is expanded into a set of round keys that will be used in the encryption rounds.
Initial Round: In the initial round, the plaintext is XORed (bitwise exclusive OR) with a round key to add an initial layer of security.
Multiple Encryption Rounds: AES-GCM performs multiple encryption rounds, each consisting of the following operations:
SubBytes: Byte-wise substitution of data using an S-box (Substitution Box).
ShiftRows: Rows of data are shifted by different amounts.
MixColumns: Columns of data are mixed to add diffusion.
AddRoundKey: A round key is XORed with the data.
Final Round: A final round is applied, which is similar to the multiple encryption rounds but omits the MixColumns operation.
Generation of Authentication Tag (MAC): AES-GCM also generates an authentication tag (Message Authentication Code) during encryption. This tag is used for data integrity and authentication.
Ciphertext and Authentication Tag: The result of the encryption process is the ciphertext (encrypted data) and the authentication tag.

AES-GCM Decryption Process:

Input: The decryption process starts with the ciphertext (the encrypted data), the secret decryption key, the IV, and the authentication tag.
Key Expansion: The secret decryption key is expanded into a set of round keys, similar to the encryption process.
Initial Round: An initial round is performed, which involves XORing the ciphertext with a round key.
Multiple Decryption Rounds: AES-GCM performs multiple decryption rounds, which are similar to the encryption rounds but in reverse order. The operations include:
Inverse ShiftRows: Reverse the row shifts.
Inverse SubBytes: Reverse the substitution using an inverse S-box.
AddRoundKey: XOR with the round key.
Final Round: The final round is performed, which is similar to the initial round in the encryption process.
Authentication Tag Verification: The received authentication tag is calculated during decryption to verify the data's integrity.
Output: If the authentication tag matches the one received, the decryption process is successful, and the plaintext is the output.

3 Project Inputs and Outputs

3.1 Input Details, Outputs – Screenshots

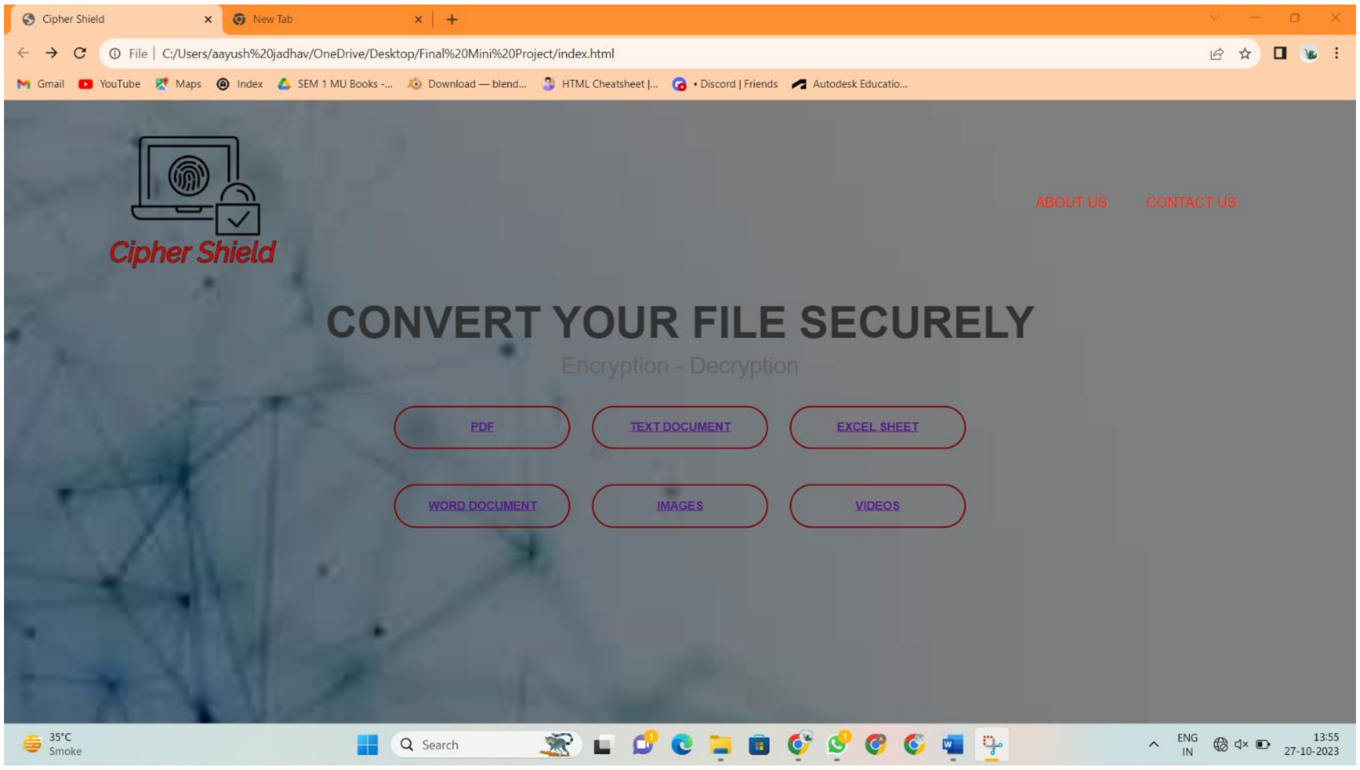
1. Input HTML Code for main webpage of Cipher Shield –

```
<!DOCTYPE html>
<html>
  <head>
    <title> Cipher Shield </title>
    <link rel="stylesheet" href="style.css">
    <style>
      /* CSS for the heading */
      .content h1 {
        font-family: Arial, sans-serif;
        font-size: 50px; /* Adjust the font size for the heading */
        font-weight: bold; /* Optionally set the font weight for the heading */
        color: #333; /* Set the text color for the heading */
      }

      /* CSS for the paragraph */
      .content p {
        font-family: "Your Paragraph Font", Arial, sans-serif; /* Specify the font for the
paragraph */
        font-size: 26px; /* Adjust the font size for the paragraph */
        font-weight: normal; /* Optionally set the font weight for the paragraph */
        color: #666; /* Set the text color for the paragraph */
      }
    </style>
  </head>
  <body>
    <div class="banner">
      <div class="navbar">
        <img src="logo.png" class="logo">
        <ul>
          <li><a href="about_us.html">ABOUT US</a></li>
          <li><a href="contact_us.html">CONTACT US</a></li>
        </ul>
      </div>

      <div class="content">
        <h1>CONVERT YOUR FILE SECURELY</h1>
        <p>Encryption - Decryption</p>

        <div>
          <button type="button"><span></span><a href="pdf.html">PDF</a></button>
          <button type="button"><span></span><a href="text.html">TEXT DOCUMENT</a></button>
          <button type="button"><span></span><a href="excel.html">EXCEL SHEET</a></button><br>
          <button type="button"><span></span><a href="word.html">WORD DOCUMENT</a></button>
          <button type="button"><span></span><a href="image.html">IMAGES</a></button>
          <button type="button"><span></span><a href="video.html">VIDEOS</a></button>
        </div>
      </div>
    </div>
  </body>
</html>
```

2. Input CSS Code for Designing main webpage of Cipher Shield –

```
*{
  margin: 0;
  padding: 0;
  font-family: sans-serif;
 }

.banner{
   width: 100%;
   height: 100vh;
   background-image: linear-gradient(rgba(0,0,0,0.50),rgba(0,0,0,0.50)),url(bg1.jpeg);
   background-size: cover;
   background-position: center;
}

.navbar{
   width: 85%;
   margin: auto;
   padding: 35px 0;
   display: flex;
   align-items: center;
   justify-content: space-between;
}

.logo{
   width: 200px;
   cursor: pointer;
}

.navbar ul li{
    list-style: none;
    display: inline-block;
    margin: 0 20px;
    position: relative;
}

.navbar ul li a{
   text-decoration: none;
   color: #ff2400;
}

.navbar ul li::after{
   content: '';
   height: 3px;
   width: 0;
   background: #810000;
   position: absolute;
   left: 0;
   bottom: -10px;
   transition: 0.5s;
}

.navbar ul li:hover::after{
   width: 100%;
}

.content{
   width: 100%;
   position: absolute;
   top: 50%;
   transform: translateY(-60%);
   text-align: center;
   color: #fff;
}

.content h1{
   font-size: 100px;
   margin-top: 50px;
}

.content{
   margin: 20px auto;
   font-weight: 100;
   line-height: 50px;
}
content p {
        font-family: "Your Chosen Font", Arial, sans-serif; /* Specify the font you want to use
*/
        font-size: 36px; /* Adjust the font size */
        font-weight: bold; /* Optionally set the font weight */
        color: #333; /* Set the text color */
      }


button{
   width: 200px;
   padding: 15px 0;
   text-align: center;
   margin: 20px 10px;
   border-radius: 25px;
   font-weight: bold;
   border: 2px solid #810000;
   background: transparent;
   color: #ff2400;
   cursor: pointer;
   position: relative;
   overflow: hidden;
}

span{
   background: #810000;
   height: 100%;
   width: 0;
   border-radius: 25px;
   position: absolute;
   left: 0;
   bottom: 0;
   z-index: -1;
   transition: 0.5s;
}

button:hover span{
   width: 100%;
}

button:hover{
   border: none:
}
```

3. Output –

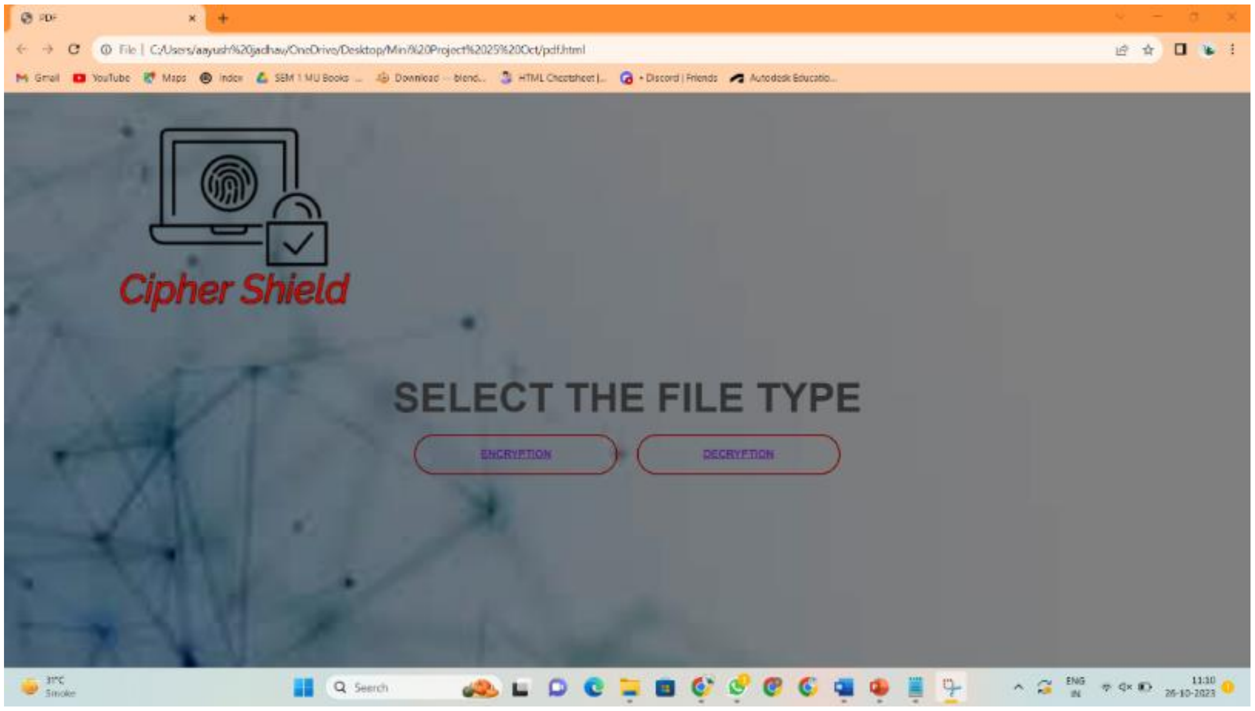


Main Webpage of Cipher Shield Website

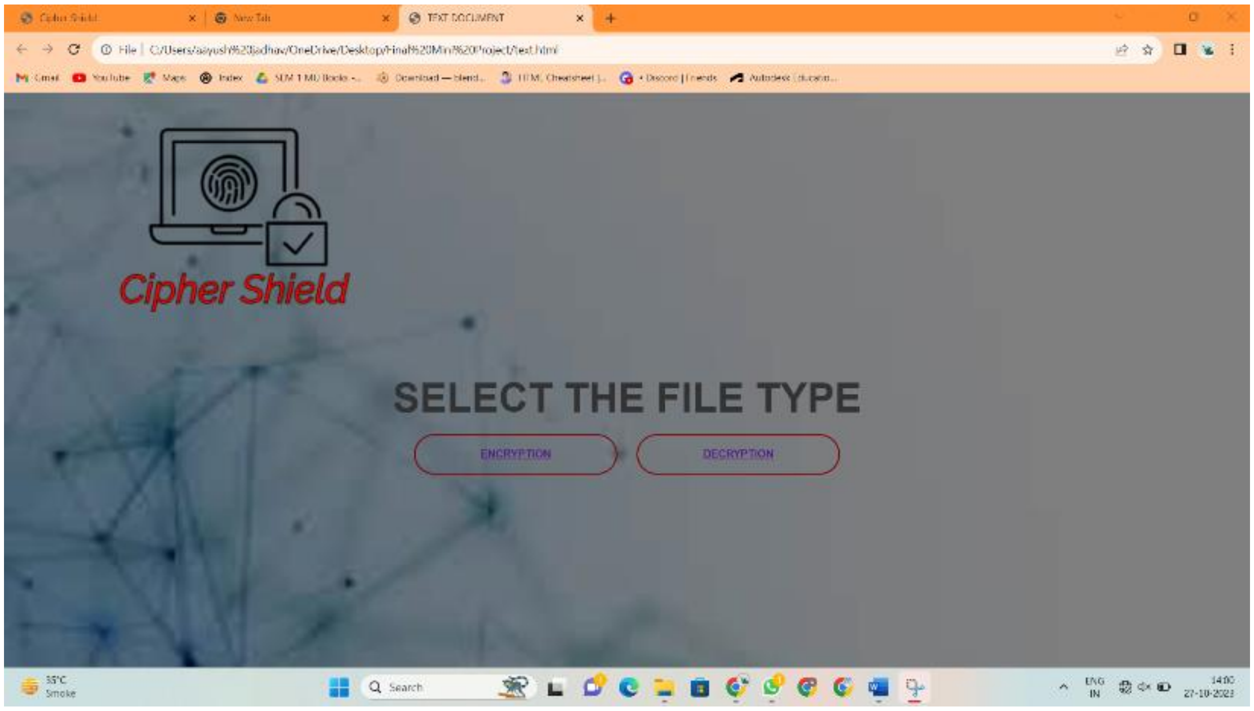
4. Input HTML Code for webpage of PDF, Text Document –

```
<!DOCTYPE html>
<html>
  <head>
    <title></title>
    <link rel="stylesheet" href="sub_style.css">
    <style>
      /* CSS for the heading */
      .content h1 {
        font-family: Arial, sans-serif;
        font-size: 50px; /* Adjust the font size for the heading */
        font-weight: bold; /* Optionally set the font weight for the heading */
        color: #333; /* Set the text color for the heading */
      }
      .navbar {
        display: flex;
        justify-content: space-between;
        align-items: center;
      }
      .logo {
        width: 300px; /* Adjust the width as needed */
        cursor: pointer;
        margin-right: auto; /* Push the logo to the left corner */
      }
    </style>
  </head>
  <body>
    <div class="banner">
      <div class="navbar">
        <img src="logo.png" class="logo">
      </div>
      <div class="content">
        <h1>SELECT THE FILE TYPE</h1>
        <div>
          <button type="button"><span></span><a
href="pdfE.html">ENCRYPTION</a></button>
          <button type="button"><span></span><a
href="pdfD.html">DECRYPTION</a></button>
        </div>
      </div>
    </div>
  </body>
</html>
```

5. Output –



Webpage of PDF Tab

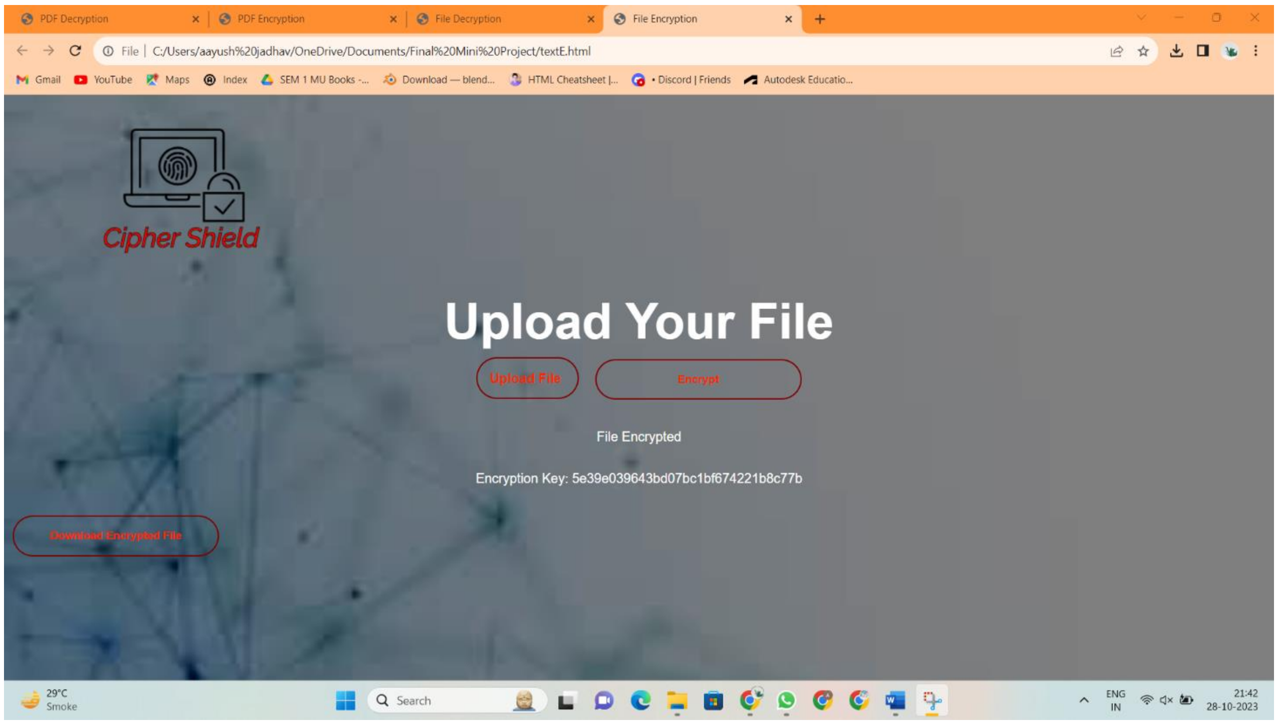


Webpage of Text Document Tab

6. Input JavaScript Code to Encrypt a Text Document –

```
<!DOCTYPE html>
<html>
<head>
    <title>File Encryption</title>
    <link rel="stylesheet" href="sub_style2.1.css">
</head>
<body>
    <div class="banner">
        <div class="navbar">
            <img src="logo.png" class="logo">
        </div>
        <div class="content">
            <h1>Upload Your File</h1>
            <label for="file-input" class="upload-button">
                <span></span>Upload File
                <input type="file" id="file-input" accept=".txt" required>
            </label>
            <button onclick="encryptFile()">Encrypt</button>
            <p id="result"></p>
            <p id="key-display" style="display: none">Encryption Key: <span id="key-
value"></span></p>
            <button id="download-button" onclick="downloadEncryptedFile()"
style="display: none; margin-left: 10px;">Download Encrypted File</button>
        </div>
    </div>
    <script>
        let encryptionKey; // Variable to store the encryption key
        let encryptedArray;

        async function encryptFile() {
            const fileInput = document.getElementById('file-input');
            const resultElement = document.getElementById('result');
            const keyDisplay = document.getElementById('key-display');
            const keyValue = document.getElementById('key-value');
            const downloadButton = document.getElementById('download-button');

            if (fileInput.files.length > 0) {
                const file = fileInput.files[0];
                const reader = new FileReader();

                reader.onload = async (e) => {
                    const fileContent = e.target.result;

                    // Generate a random encryption key
                    encryptionKey = new Uint8Array(16); // 128 bits (16 bytes)
                    crypto.getRandomValues(encryptionKey);

                    // Convert the key to a hexadecimal string for display
                    const keyString = Array.from(encryptionKey).map(byte =>
byte.toString(16).padStart(2, '0')).join('');

                    // Import the key as a CryptoKey
                    const importedKey = await crypto.subtle.importKey(
                        'raw', // Key format
                        encryptionKey,
                        'AES-GCM',
                        true, // Extractable
                        ['encrypt']
                    );

                    // Encode the file content as a base64 string
                    const encodedData = btoa(fileContent);

                    // Encrypt the file using the Web Crypto API
                    const encryptedData = await crypto.subtle.encrypt(
                        { name: 'AES-GCM', iv: encryptionKey },
                        importedKey, // Use the imported key
                        new TextEncoder().encode(encodedData)
                    );

                    encryptedArray = new Uint8Array(encryptedData);

                    resultElement.textContent = 'File Encrypted';
                    keyDisplay.style.display = 'block';
                    keyValue.textContent = keyString;
                    downloadButton.style.display = 'block';
                };

                reader.readAsText(file);
            } else {
                resultElement.textContent = 'Please select a file.';
                keyDisplay.style.display = 'none';
                downloadButton.style.display = 'none';
            }
        }

        function downloadEncryptedFile() {
            if (encryptedArray) {
                // Replace 'encrypted_file.enc' with your desired file name
                const fileName = 'encrypted_file.enc';

                // Create a Blob with the encrypted data
                const encryptedBlob = new Blob([encryptedArray], { type: 
'application/octet-stream' });

                // Create a link and trigger the download
                const a = document.createElement('a');
                a.href = URL.createObjectURL(encryptedBlob);
                a.download = fileName;
                a.click();
            }
        }
    </script>
</body>
</html>
```

7. Output –


Upload Your File

File Encrypted

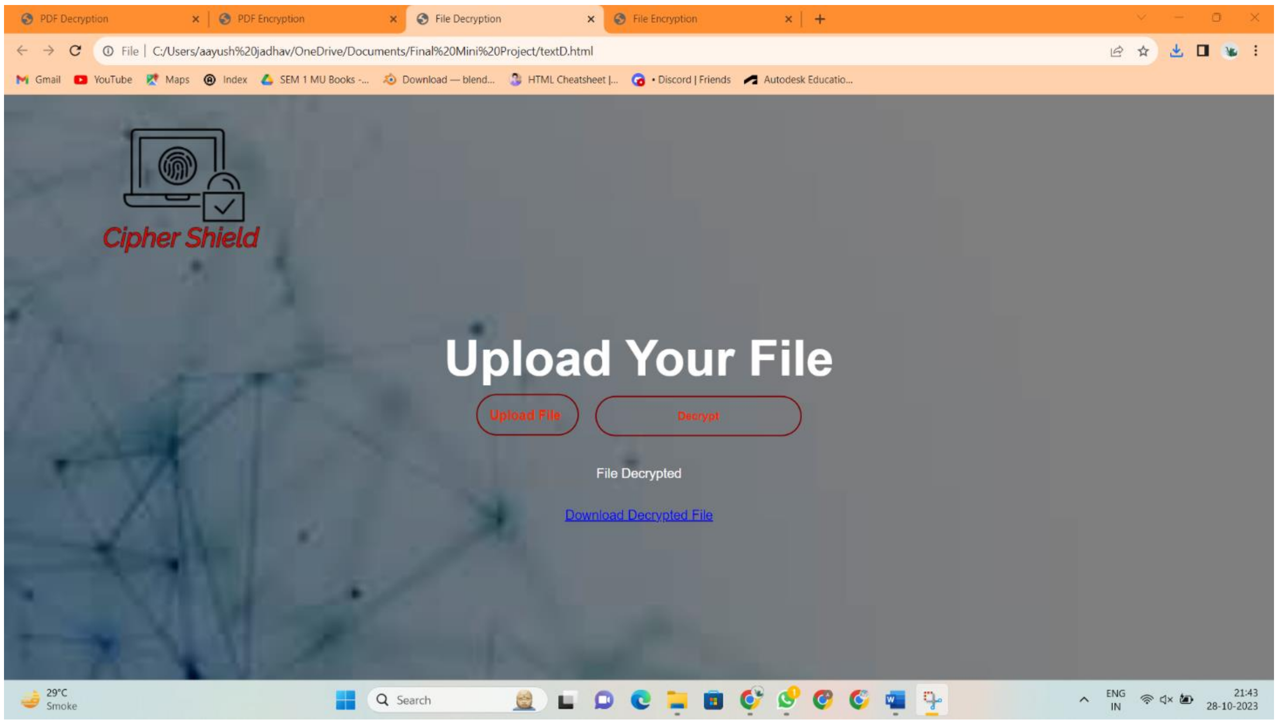
Encryption Key: 5e39e039643bd07bc1bf674221b8c77b

Text Document tab after encrypting the text file

8. Input JavaScript Code to Decrypt a Text Document –

```
<!DOCTYPE html>
<html>
<head>
    <title>File Decryption</title>
    <link rel="stylesheet" href="sub_style2.1.css">
</head>
<body>
    <div class="banner">
      <div class="navbar">
        <img src="logo.png" class="logo">
    </div>
    <div class="content">
    <h1>Upload Your File</h1>
    <label for="file-input" class="upload-button">
      <span></span>Upload File
      <input type="file" id="file-input" accept=".enc" required>
    </label>
    <button onclick="decryptFile()">Decrypt</button>
    <p id="result"></p>
    <p id="key-display" style="display: none">Encryption Key: <span id="key-
value"></span></p>
    <a id="download-link" style="display: none" download="decrypted_file.txt">Download
Decrypted File</a>
   </div>

    <script>
        let decryptionKey; // Variable to store the decryption key

        async function decryptFile() {
            const fileInput = document.getElementById('file-input');
            const resultElement = document.getElementById('result');
            const keyDisplay = document.getElementById('key-display');
            const keyValue = document.getElementById('key-value');
            const downloadLink = document.getElementById('download-link');

            if (fileInput.files.length > 0) {
                const file = fileInput.files[0];
                const reader = new FileReader();

                reader.onload = async (e) => {
                    const fileContent = e.target.result;

                    const key = prompt('Enter the encryption key (32 hexadecimal
characters):');
                    if (!key || key.length !== 32) {
                        return;
                    }

                    try {
                        // Convert the key from a hexadecimal string to a Uint8Array
                        const keyArray = new Uint8Array(16);
                        for (let i = 0; i < 16; i++) {
                            keyArray[i] = parseInt(key.substr(i * 2, 2), 16);
                        }

                        // Import the key as a CryptoKey
                        const importedKey = await crypto.subtle.importKey(
                            'raw', // Key format
                            keyArray,
                            'AES-GCM',
                            true, // Extractable
                            ['decrypt']
                        );

                        // Decrypt the file using the Web Crypto API
                        const decryptedData = await crypto.subtle.decrypt(
                            { name: 'AES-GCM', iv: keyArray },
                            importedKey, // Use the imported key
                            new Uint8Array(fileContent)
                        );

                        const decryptedText = new TextDecoder().decode(decryptedData);

                        // Decode the decrypted base64 data
                        const decodedData = atob(decryptedText);

                        resultElement.textContent = 'File Decrypted';
                        keyDisplay.style.display = 'none';
                        keyValue.textContent = ''; // Clear the displayed key
                        const decryptedBlob = new Blob([decodedData], { type:
'text/plain' });
                        downloadLink.href = URL.createObjectURL(decryptedBlob);
                        downloadLink.style.display = 'block';
                    } catch (error) {
                        resultElement.textContent = 'Decryption failed. Please check
the key.';
                        keyDisplay.style.display = 'none';
                        keyValue.textContent = ''; // Clear the displayed key
                        downloadLink.style.display = 'none';
                    }
                };

                reader.readAsArrayBuffer(file);
            } else {
                resultElement.textContent = 'Please select a file.';
                keyDisplay.style.display = 'none';
                keyValue.textContent = ''; // Clear the displayed key
                downloadLink.style.display = 'none';
            }
        }
    </script>
</body>
</html>
```

9. Output –

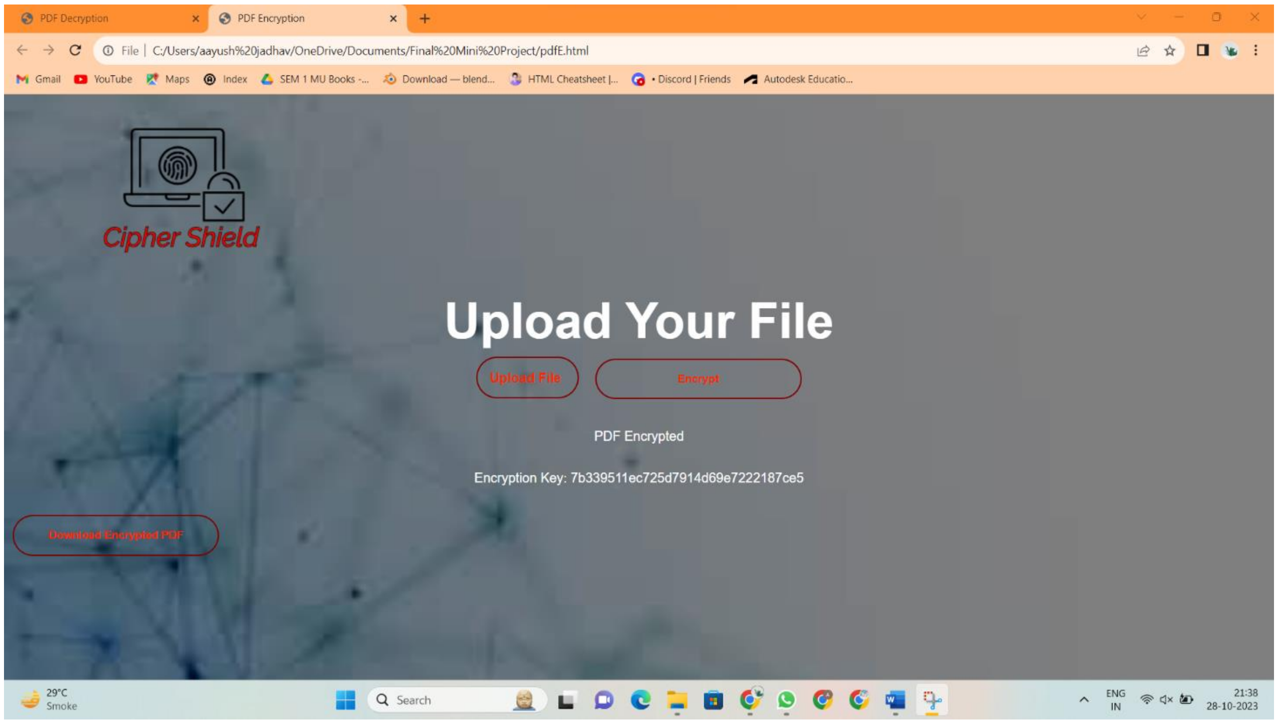


Text Document tab after decrypting the text file

10.Input JavaScript Code to Encrypt a PDF –

```
<!DOCTYPE html>
<html>
<head>
    <title>PDF Encryption</title>
    <link rel="stylesheet" href="sub_style2.1.css">
</head>
<body>
    <div class="banner">
        <div class="navbar">
            <img src="logo.png" class="logo">
        </div>
        <div class="content">
            <h1>Upload Your File</h1>
            <label for="file-input" class="upload-button">
                <span></span>Upload File
                <input type="file" id="file-input" accept=".pdf" required>
            </label>
            <button id="encrypt-button" onclick="encryptPDF()">Encrypt</button>
            <p id="result"></p>
            <p id="key-display" style="display: none">Encryption Key: <span id="key-
value"></span></p>
            <button id="download-button" onclick="downloadEncryptedPDF()"
style="display: none; margin-left: 10px;">Download Encrypted PDF</button>
        </div>
    </div>

    <script>
        let encryptionKey; // Variable to store the encryption key
        let encryptedArray; // Variable to store encrypted data

        async function encryptPDF() {
            const fileInput = document.getElementById('file-input');
            const resultElement = document.getElementById('result');
            const keyDisplay = document.getElementById('key-display');
            const keyValue = document.getElementById('key-value');
            const downloadButton = document.getElementById('download-button');

            if (fileInput.files.length > 0) {
                const file = fileInput.files[0];
                const reader = new FileReader();

                reader.onload = async (e) => {
                    const fileContent = e.target.result;

                    // Generate a random encryption key
                    encryptionKey = new Uint8Array(16); // 128 bits (16 bytes)
                    crypto.getRandomValues(encryptionKey);

                    // Convert the key to a hexadecimal string for display
                    const keyString = Array.from(encryptionKey)
                        .map(byte => byte.toString(16).padStart(2, '0'))
                        .join('');
                    // Import the key as a CryptoKey
                    const importedKey = await crypto.subtle.importKey(
                        'raw', // Key format
                        encryptionKey,
                        'AES-GCM',
                        true, // Extractable
                        ['encrypt']
                    );

                    // Encrypt the PDF file using the Web Crypto API
                    const encryptedData = await crypto.subtle.encrypt(
                        { name: 'AES-GCM', iv: encryptionKey },
                        importedKey, // Use the imported key
                        new Uint8Array(fileContent)
                    );

                    encryptedArray = new Uint8Array(encryptedData); // Store encrypted
data

                    resultElement.textContent = 'PDF Encrypted';
                    keyDisplay.style.display = 'block';
                    keyValue.textContent = keyString;
                    downloadButton.style.display = 'block';
                };

                reader.readAsArrayBuffer(file);
            } else {
                resultElement.textContent = 'Please select a PDF file.';
                keyDisplay.style.display = 'none';
                downloadButton.style.display = 'none';
            }
        }

        function downloadEncryptedPDF() {
            if (encryptionKey) {
                // Replace 'encrypted_file.enc' with your desired file name
                const fileName = 'encrypted_file.enc';

                // Create a Blob with the encrypted data
                const encryptedBlob = new Blob([encryptedArray], { type:
'application/octet-stream' });

                // Create a link and trigger the download
                const a = document.createElement('a');
                a.href = URL.createObjectURL(encryptedBlob);
                a.download = fileName;
                a.click();
            }
        }
    </script>
</body>
</html>
```

11.Output –



Upload Your File

PDF Encrypted

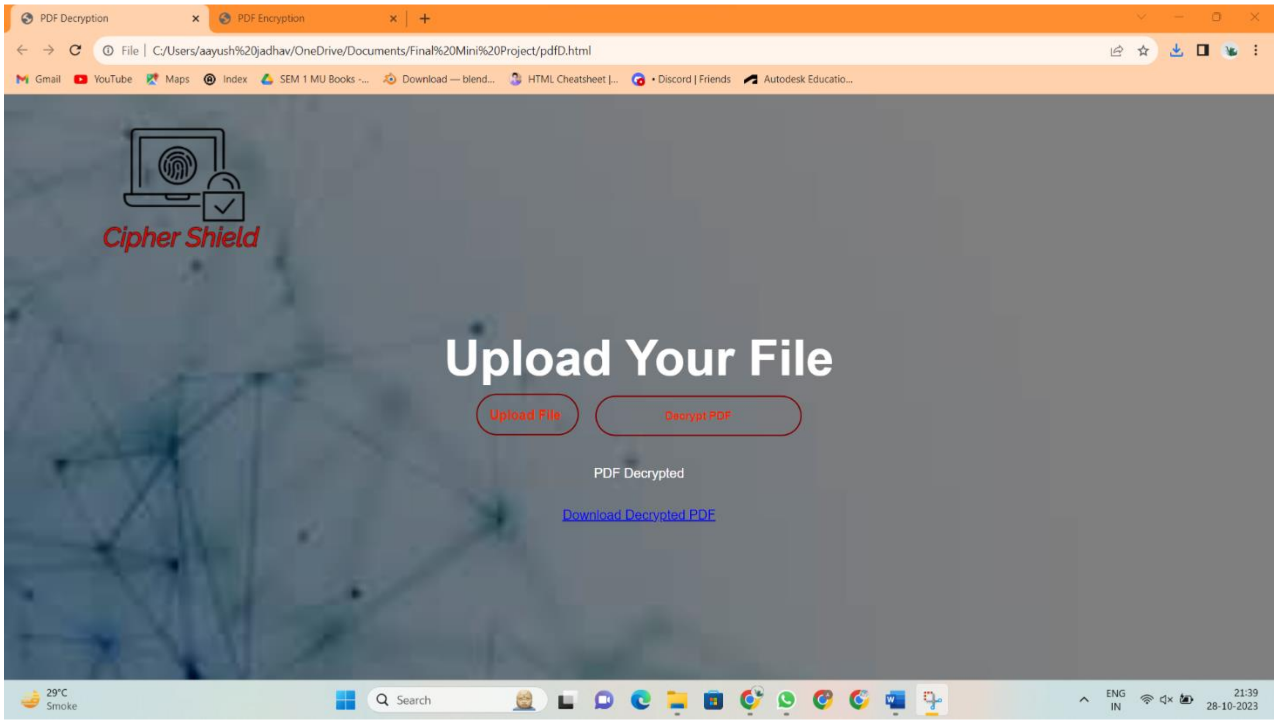
Encryption Key: 7b339511ec725d7914d69e7222187ce5

PDF tab after encrypting the pdf

12.Input JavaScript Code to Decrypt a PDF –

```
<!DOCTYPE html>
<html>
<head>
  <title>PDF Decryption</title>
  <link rel="stylesheet" href="sub_style2.1.css">
</head>
<body>
  <div class="banner">
    <div class="navbar">
      <img src="logo.png" class="logo">
    </div>
  <div class="content">
  <h1>Upload Your File</h1>
  <label for="file-input" class="upload-button">
        <span></span>Upload File
        <input type="file" id="file-input" accept=".enc" required>
  </label>

  <button onclick="decryptPDF()">Decrypt PDF</button>
  <p id="result"></p>
  <p id="key-display" style="display: none">Encryption Key: <span id="key-value"></span></p>
  <a id="download-link" style="display: none" download="decrypted_file.pdf">Download Decrypted PDF</a>
 </div>
  <script>
    let decryptionKey; // Variable to store the decryption key

    async function decryptPDF() {
      const fileInput = document.getElementById('file-input');
      const resultElement = document.getElementById('result');
      const keyDisplay = document.getElementById('key-display');
      const keyValue = document.getElementById('key-value');
      const downloadLink = document.getElementById('download-link');

      if (fileInput.files.length > 0) {
        const file = fileInput.files[0];
        const reader = new FileReader();

        reader.onload = async (e) => {
          const fileContent = e.target.result;

          const key = prompt('Enter the encryption key (32 hexadecimal characters):');
          if (!key || key.length !== 32) {
            return;
          }

          try {
            // Convert the key from a hexadecimal string to a Uint8Array
            const keyArray = new Uint8Array(16);
            for (let i = 0; i < 16; i++) {
              keyArray[i] = parseInt(key.substr(i * 2, 2), 16);
            }

            // Import the key as a CryptoKey
            const importedKey = await crypto.subtle.importKey(
              'raw', // Key format
              keyArray,
              'AES-GCM',
              true, // Extractable
              ['decrypt']
            );

            // Decrypt the file using the Web Crypto API
            const decryptedData = await crypto.subtle.decrypt(
              { name: 'AES-GCM', iv: keyArray },
              importedKey, // Use the imported key
              new Uint8Array(fileContent)
            );

            // Create a Blob for the decrypted PDF
            const decryptedBlob = new Blob([decryptedData], { type: 'application/pdf' });

            resultElement.textContent = 'PDF Decrypted';
            keyDisplay.style.display = 'none';
            keyValue.textContent = ''; // Clear the displayed key
            downloadLink.href = URL.createObjectURL(decryptedBlob);
            downloadLink.style.display = 'block';
          } catch (error) {
            resultElement.textContent = 'Decryption failed. Please check the key.';
            keyDisplay.style.display = 'none';
            keyValue.textContent = ''; // Clear the displayed key
            downloadLink.style.display = 'none';
          }
        };

        reader.readAsArrayBuffer(file);
      } else {
        resultElement.textContent = 'Please select an encrypted PDF file.';
        keyDisplay.style.display = 'none';
        keyValue.textContent = ''; // Clear the displayed key
        downloadLink.style.display = 'none';
      }
    }
  </script>
</body>
</html>
```

13.Output –



PDF tab after decrypting the pdf

4. Conclusion and Future Scope

4.1 Conclusion

When it comes to encryption and decryption, it’s important to understand the difference between the two.

Encryption is the process of transforming readable data into an unreadable format, while Decryption is the process of transforming unreadable data into readable format.

Encryption is used to protect data from unauthorized access, while Decryption is used to restore data to its original format. Encryption and Decryption are essential for ensuring the security of data.

The File Encryption & Decryption mini-project conclude with successful output, providing user a reliable & user-friendly system to convert plain text into cipher text and vice-versa.

Cipher shield provide one stop solution to convert all types of your file in one go whether it is plain text ,word file ,image file, video file, Excel sheet. Cipher Shield can Decrypt and Encrypt all types of your file in a seemless and easy way

4.2 Future Scope

a. **Stronger Encryption Algorithms:** In the future, you can expand the project by adding support for more advanced encryption algorithms, including post-quantum cryptography.
b. **Cloud Integration:** Extend the project to support encryption and decryption of files stored in cloud services like Dropbox or Google Drive.
c. **Mobile Application:** Develop a mobile version of the application, allowing users to encrypt and decrypt files on their smartphones securely.
d. **Biometric Authentication:** Implement biometric authentication methods (e.g., fingerprint or facial recognition) to enhance security.
e. **Blockchain Integration:** Explore how blockchain technology can be used for secure key management and decentralized authentication.
f. **Multi-Platform Compatibility:** Make the application compatible with multiple operating systems (Windows, macOS, Linux) for broader usability.
g. **File Compression:** Add file compression features to save space before encryption, improving efficiency and reducing storage costs.
h. **Ransomware Detection:** Incorporate ransomware detection and prevention mechanisms to safeguard encrypted files against malicious attacks.
i. **Performance Optimization:** Optimize the project for speed and efficiency, especially when working with large files.
j. **Machine Learning for Threat Detection:** Implement machine learning models to detect potential threats and anomalies in encrypted files.

5. Reference

Here is list of references used for making of mini project :

- "Cryptography and Network Security" by William Stallings (book). The book provided overall idea on the process of file encryption and decryption.
- Cryptography tutorials and articles on websites like GeeksforGeeks.
- Open-source encryption and decryption projects on platforms like GitHub for code examples and inspiration.
- YouTube videos on JavaScript code for File Encryption and Decryption.
- Various websites like Skiff, Stack Overflow, GitHub and Tutorialzine to understand the working of JavaScript code on File Encryption and Decryption.
- Logomaker to make various logos used in the Cipher Shield website.
- Extensive use of ChatGPT for various tech and literary supports (codes, information, instructions and reports)
- Use of You.AI to understand working and process of various file encryption and decryption algorithms.
- Websites like Studou, ResearchGate, Scribd to understand various sub-topics of mini project report.
- **IEEE**'s (Institute of Electrical and Electronics Engineers) detail Research report on a Normal File Encryption and Decryption