National College of Ireland

BSCH In Computing

Cyber Security

2022/2023

Nathan Andrews

X19135394

X19135394@student.ncirl.ie

Salt N Pepper

Technical Report

Contents

Executive Summary

This documentations purpose is to give a deep understanding of the inner workings, motives, users needs, and functionality of the application as a whole.

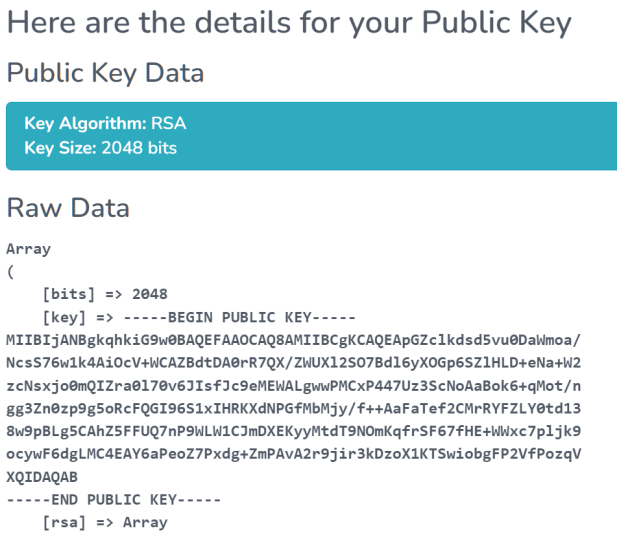
The user should have Google Chrome installed. Then you can simply follow this link >Salt N Pepper< This is a link directly to the Chrome store, from here the user only has to click add to Chrome to install the application.

The purpose of Salt N Pepper is to provide the user with an easy-to-use application where they have the power to control their own encryption. The idea is to offer a high-level encryption standard to users without much technical knowledge.

Salt N Pepper allows users to generate RSA Public and Private key pairs. Using an online decoder we can verify the integrity of the key pairs as shown here on the right. Any user can verify the integrity of their keys and be assured they are using the proper standards.

Key Decoder Link: Click here (Paste details of public key)

The rest of the document will go further into some more technical aspects of the application, including snapshots of encryption code. There will be a follow-up link in that section to GitHub instead of cluttering the document.



Here are the details for your Public Key

Public Key Data

Key Algorithm: RSA
Key Size: 2048 bits

Raw Data

```
Array
(
    [bits] => 2048
    [key] => -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEApGZclkdsd5vu0DaWmoa/
NcsS76w1k4AiOcV+WCAZBdtDA0rR7QX/ZWUX12SO7Bdl6yXOGp6SZlHLD+eNa+W2
zcNsxjo0mQIZra0l70v6JIsfJc9eMEWALgwwPMCxP447Uz3ScNoAaBok6+qMot/n
gg3Zn0zp9g5oRcFQGI96S1xIHRKXdNPGfMbMjy/f++AaFaTef2CMrRYFZLY0td13
8w9pBLg5CAhZ5FFUQ7nP9WLW1CJmDXEKyyMtdT9NOmKqfrSF67fHE+WWxc7pljk9
ocywF6dgLMC4EAY6aPeoZ7Pxdg+ZmPAvA2r9jir3kDzoX1KTSwiobgFP2VfPozqV
XQIDAQAB
-----END PUBLIC KEY-----
    [rsa] => Array
```

1.1. Background

The idea for Salt N Pepper came because of the fact that many people use services that are encrypted, but are not necessarily in control of that encryption themselves. This was to give users more control over their own data which they would like to encrypt privately. This will allow users to generate their own keys and control the encryption they use, and they can be assured that it Is safe as they have the key for decryption.

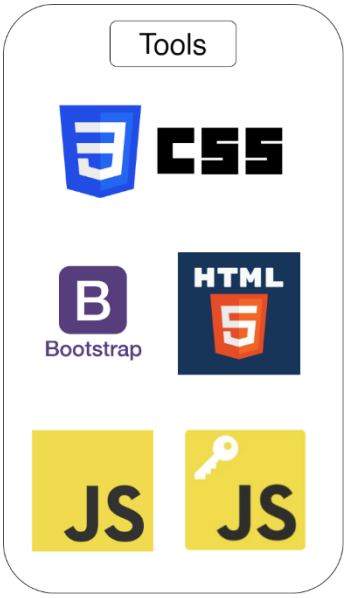
1.2. Aims

The project aims to achieve a high level of encryption.

AES Advanced Encryption Standard 256 BIt is one of the highest forms of encryption standard available. This is a symmetric type of encryption that will offer ease of use for the user. It acts like a password but on a binary level, the file is protected.

RSA Public Key Encryption, the application aims to use a hybrid version of AES and RSA. This is because of the constraints of RSA to encrypt large data. It is more efficient to encrypt large data with AES and then encrypt the AES encryption key with RSA Public key encryption. This also gives 2 levels of encryption increasing the security to an even higher standard.

1.3. Technology



The project will be integrated into Google Chrome as an extension the tools that will be available for the job are HTML, JavaScript, and CSS. Very much the same tools that are available for Web Design.

CrytoJS will be the encryption framework that I will use to implement the encryption standards.

HTML will be the format and layout of the pages itself.

Bootstrap I will use for styling which consists of CSS.

Node.js Selenium, and Mocha for testing the functionality of the application. Mocha is a testing frame for Javascript to enable to use of testing syntax. We can then also integrate selenium to run the application in the browser.

This project purposely avoids the use of a database. This is to give more security to the user knowing that there's no information regarding their files or keys being stored anywhere by third parties.

1.4. Structure

Functional Requirements: This section will provide the details as to what the user should benefit from using the application. Use case diagrams to show interactions between the user and the application. With further use case description providing information on further details.

User Requirements: This section should demonstrate in terms that the user can understand the functionality and requirements of the user and what can be expected of the application.

Design & Architecture: This section will give a brief demonstration of the design and architecture that is going to be used in the development of the application.

Implementation: This will provide a more detailed view of how the implementation of the product will be done with code snippets of already completed project code. Using the tools that are set out in the Design and Architecture

Graphical User Interface: This section illustrates the user interface of the application, showing the navigation and the different pages of the application. There are many help options and descriptions available to help the user fully understand the concepts and how to use the application.

Testing: This section will go into the testing methods that were used in the project to establish a more robust and professional product.

Evaluation: This section will go into the performance of AES encryption and other standards that could be used.

Conclusions: This will consider the advantages and disadvantages of the project overall.

2.0 System

2.1. Requirements

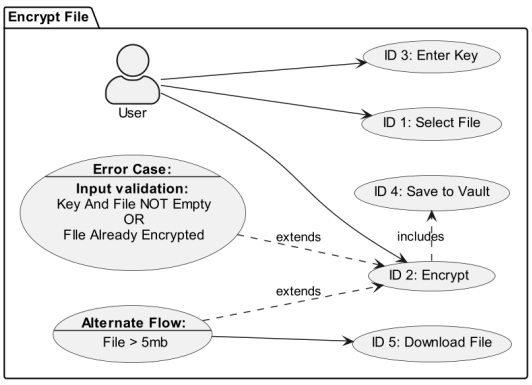
Users would not require much more than 30 mins of training to be able to operate the application. The application takes a lot of the work and abstracts it from the user. The user should be able to write text in whatever language they use, have entry-level efficiency in operating a computer and select files and navigate the file system and have a Google Chrome web browser installed.

2.1.1. Functional Requirements

The list contains the functional requirements that will be implemented for Salt N Pepper. Each requirement has a priority that is listed in the table below:

Priority	**Description**
Priority 1	Top Priority – Critical Functionality
Priority 2	High Priority – Core Functionality
Priority 3	Med(ium) Priority – Main Functionality
Priority 4	Low Priority – Requested Functionality

2.1.1.1. Use Case Diagram



2.1.1.2. Requirement Encrypt File

User will have to have downloaded the application

2.1.1.3. Description & Priority

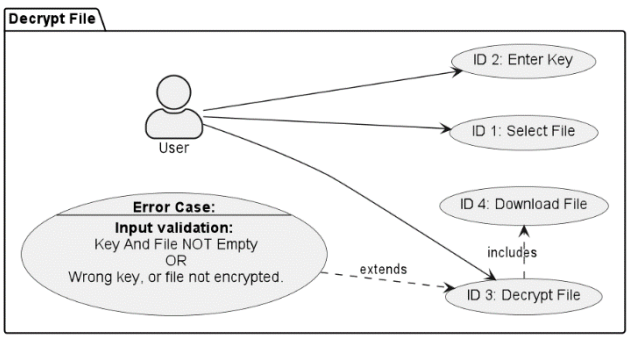
Top Priority

This is the main functionality of the application. Giving the user the ability to enter their own key and know that AES encryption standard is being used.

2.1.1.4. Use Case

Use Case ID:	UC-1
Use Case Nam	Encrypt File AES
Actors, Primary, Secondary:	User
Triggers:	The user will select the AES tab from the application and will select a file and enter a key then select encrypt.
Pre-Condition:	1: The user will have to navigate to the AES tab of the application.
Post-Condition:	1: A download for the file will begin
Normal Flow:	1: The user will have to select a file, 2: The user will have to enter an encryption key in the key field, 3: The user will have to click Encrypt
Alternate Flows:	1: File Greater than 5 MB download instead of saving to the vault. (Will have to pay for a storage solution for greater storage, 5 MB is a free cap)
Exceptional Flows:	1: File already encrypted, 2: Input Validation 3: If file greater than 5 mb download instead of saving to vault
Includes:	1: Save to vault, this will save a reference of the file and key to the vault tab
Extends:	None

2.1.1.5 Use Case Diagram



2.1.1.6 Description & Priority

Top Priority

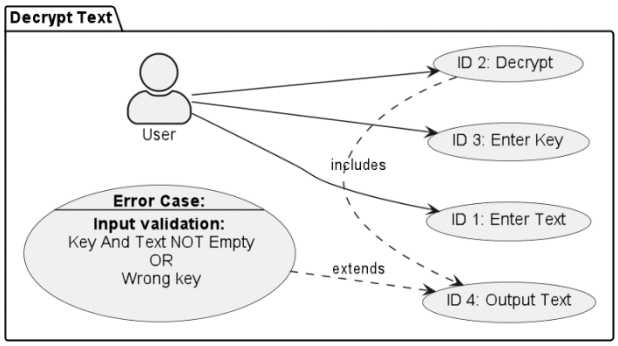
This functionality is required as a part of the encryption. Without being able to decrypt the file the encryption wouldn't be very useful. This is to be developed alongside the encryption functionality.

2.1.1.7 Use Case

Use Case ID:	UC-2
Use Case Name:	Decrypt File
Actors, Primary, Secondary:	User
Triggers:	The user will select AES tab from the application and will select a file and enter a key then select decrypt.
Pre-Condition:	1: The user will have to navigate to the AES tab of the application.

Post-Condition:	1: A download for the file will begin
Normal Flow:	1: The user will have to select a file, 2: The user will have to enter a decryption key in the key field, 3: The user will have to click decrypt
Alternate Flows:	None
Exceptional Flows:	1: File Not encrypted, 2: Wrong decryption key 3: Input Validation
Includes:	1: Download File
Extends:	None

2.1.1.8 Use Case Diagram



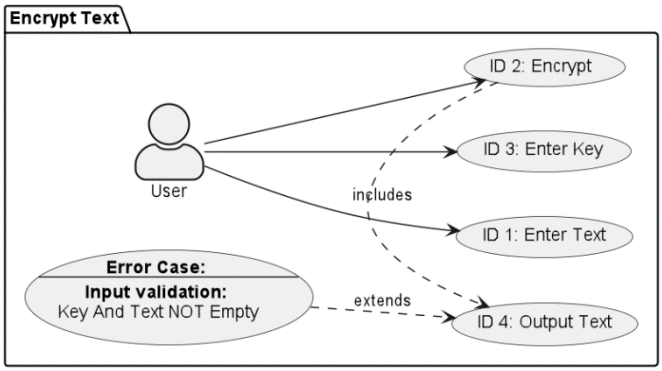
2.1.1.9 Description & Priority

Priority: Medium

This functionality is for encrypting a single text input. The user will input text into a field, and it will output the encrypted text message back to the user where they can then copy that text and send it. This is not the main part of the application and priority is medium–low

2.1.1.10 Use Case

Use Case ID:	UC-3
Use Case Name:	Decrypt Text
Actors, Primary, Secondary:	User
Triggers:	User will select decrypt on the AES page when text input filled, and key filled.
Pre-Condition:	1: The user will have to navigate to the AES tab of the application.
Post-Condition:	1: A new element will inject into the DOM and show the decrypted message in a text area
Normal Flow:	1: The user will enter text into the form labeled text 2: The user will enter a key into the form row labeled Decryption Key 3: The user will then select decrypt 4: The deciphered message will be output into the app
Alternate Flows:	None
Exceptional Flows:	1: Input validation for text fields and key 2: Wrong Key entered
Includes:	1: Output text
Extends:	None



2.1.1.11 Use Case Diagram

2.1.1.12 Description & Priority

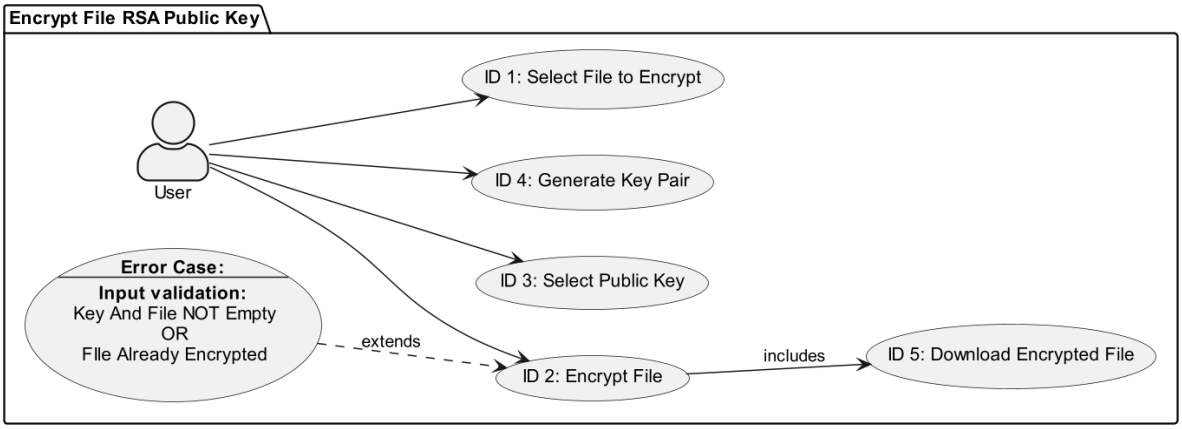
Priority: Medium

This use case will Output the hashed encrypted message to the user which they can then copy and decrypt or send to another user. This will demonstrate to the user what is happening with encryption.

2.1.1.13 Use Case

Use Case ID:	UC-4
Use Case Name:	Encrypt Text
Actors, Primary, Secondary:	User
Triggers:	The user will select Encrypt on the AES page when text input is filled, and the key is filled.
Pre-Condition:	1: The user will have to navigate to the AES tab of the application.
Post-Condition:	1: A new element will be created in the DOM that will output the encrypted hashed message
Normal Flow:	1: The user will enter text into the form labeled text 2: The user will enter a key into the form row labeled Decryption Key 3: The user will then select Encrypt 4: Encrypted hashed message will be output to the app
Alternate Flows:	None
Exceptional Flows:	1: Input validation for text fields and key
Includes:	1: Output text
Extends:	None

2.1.1.14 Use Case Diagram



2.1.1.15 Description & Priority

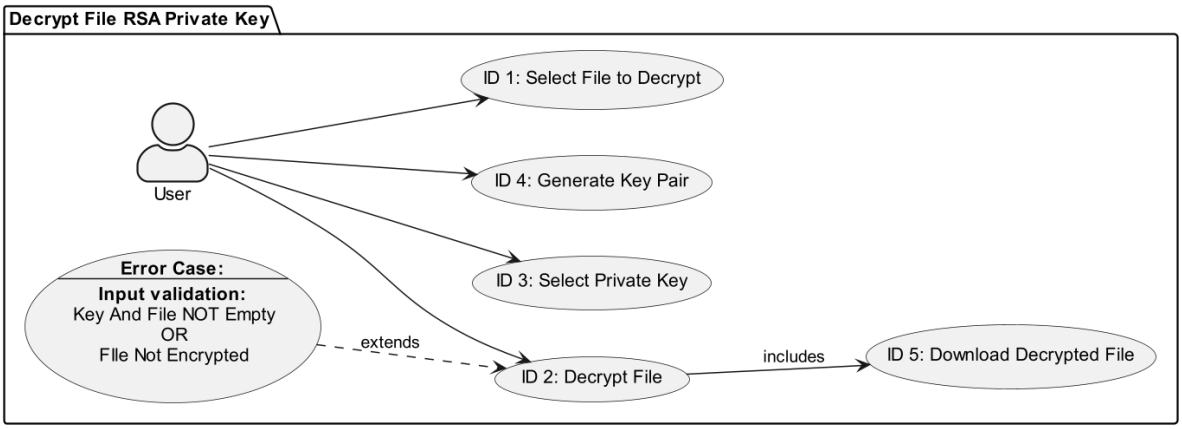
Priority: High

This use case will demonstrate the flow of the RSA Public key encryption. This assumes that the user would have already generated a key pair and kept the keys safe and are accessible on their system.

2.1.1.16 Use Case

Use Case ID:	UC-5
Use Case Name:	Encrypt File RSA Public/Private Key
Actors, Primary, Secondary:	User
Triggers:	The user will select Encrypt on the RSA page when a file is selected for encryption and a public key is selected as a key
Pre-Condition:	1: The user will have to navigate to the RSA tab of the application.
Post-Condition:	1: A download will begin for the encrypted file
Normal Flow:	1: The user will select their public key 2: The user will select the file to encrypt 3: The user will press the encrypt button 4: A download will begin for the encrypted file
Alternate Flows:	None
Exceptional Flows:	1: Input validation for the file to encrypt and public key
Includes:	1: Download File
Extends:	None

2.1.1.17 Use Case Diagram



2.1.1.18 Description & Priority

Priority: High

This use case will demonstrate the flow of the RSA Private key decryption. This assumes that the user would have already generated a key pair and kept the keys safe and accessible on their system.

2.1.1.19 Use Case

Use Case ID:	UC-6

Use Case Name:	Decrypt File RSA Public/Private Key
Actors, Primary, Secondary:	User
Triggers:	The user will select Decrypt on the RSA page when a file is selected for encryption and a private key is selected as a key
Pre-Condition:	1: The user will have to navigate to the RSA tab of the application.
Post-Condition:	1: A download will be commenced for the decrypted file
Normal Flow:	1: The user will select their private key 2: The user will select the file to decrypt 3: The user will press the decrypt button 4: A download will begin for the decrypted file
Alternate Flows:	None
Exceptional Flows:	1: Input validation for private key and file to decrypt
Includes:	1: Download FIle
Extends:	None

2.1.2. User Requirements

Users will have to have Google Chrome web browser installed and be able to install extensions. Using Salt N Pepper the user will be able to Encrypt and Decrypt messages with ease. They simply enter a message and a key or in other words a password for the file and click encrypt. They can be reassured by using the latest standards of encryption used by the FBI known as AES that they are in safe hands. No information is being stored about the files or encryption keys they are using and no server connection or database is used. Completely isolated to their system.

The user can also encrypt and decrypt any flies of their choosing, the most popular are .txt, .zip, and image files also.

The interface of SnP will be intuitive and minimalistic. The user will have full control over their own encryption. There will be an info page with some information about how encryption works to help the user to learn about the safety of encryption and encourage more users to participate.

2.1.3. Data Requirements

There shouldn't be any data requirements for this project. One of the aims of Salt N Pepper is to avoid the storing of information of the user. The application will not have any server or database connected to it.

2.1.4. Usability Requirements

The idea of Salt N Pepper is to abstract complexity from the user enabling them to use RSA Public/Private key encryption and also AES (Advanced Encryption Standard) at ease.

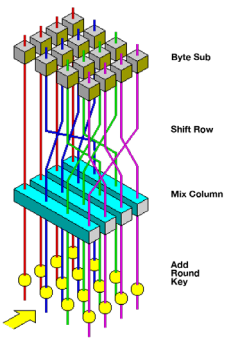
RSA Public Key encryption will allow the user to generate a Public and Private key pair they can use for their own needs.

Salt N Pepper will use a dynamic UI and keep only the required information on the screen at any given time. If the user requires more information it will be displayed for only the time it is required and updated accordingly hence dynamic.

The application solves the problem of encrypting large files or data with RSA. RSA is known to be inefficient at encrypting large pieces of data. The workaround for this problem is to randomly generate a key and then encrypt it with AES. The encryption key used will then be encrypted with RSA public and private keys. This allows for efficient encryption and a higher level of security.

The system will provide directions and clear instructions to the user regarding inputs, errors, and general help. There will be help buttons to further explain what encryption is, how to use the application, what the application does, and how the user is protected.

2.2. Design & Architecture



JavaScript and CryptoJS Encryption Framework are the main design implementations in this project. CryptoJS provides a large framework to use all types of encryption standards.

CrytoJS is a renowned framework that is continuously growing and follows best practices in implementing its encryption tools.

Bootstrap is used for the styling and CSS of the application to help with responsiveness and enhance the overall quality of the product.

HTML Hyper Text Markup Language is the base for the application. The pages you see will be written in HTML and are also prone to the same vulnerability as other web applications.

JavaScript will be used to create logic and control elements of the application. This will allow me to code like Java. Google also has the ability to use the Google Web Toolkit to convert Java code into JavaScript

Selenium is a popular open-source framework used for automated web testing. The main purpose of selenium is to test web applications to ensure that they work as expected and is a key tool in test-driven or behavior-driver development.

Mocha, is a feature-rich JS test framework that is running on node.js and in the browser. it provides a flexible user-friendly platform for setting up tests in Java. Mocha provides “hooks” such as “describe()”, “context”, “before()”, “beforeEach” and many more hooks that are used to provide structure to tests.

2.3. Implementation

(Project Code)
Encrypt File AES:

```
function encryptFile() {
  var file = document.getElementById( elementId: "file").files[0];
  if(file.name.includes( value: ".encrypted")) {
    alert("File already encrypted.");
    return;
  }
    var key = document.getElementById( elementId: "key").value;
    var reader = new FileReader();
    reader.onload = function (e : ProgressEvent<FileReader> ) {
    var fileData = e.target.result;
    var encrypted = CryptoJS.AES.encrypt(fileData, key);

    //Saving the encrypted file to file storage
    try {
      localStorage.setItem(file.name, encrypted);
    } catch (QuotaExceededError) {
      alert("File cannot exceed 5mb, File not saved into storage. Downloading instead.");
      var link = document.getElementById( elementId: "download");
      link.setAttribute( qualifiedName: "href", value: "data:application/octet-stream," + encrypted);
      link.setAttribute( qualifiedName: "download", value: file.name + ".encrypted");
      link.click();
      return;
    }
```

```
    alert("file saved to storage");
  }
  reader.readAsDataURL(file);
}
```

The above is an example of the code used for encrypting the file using AES. A very similar function is being used for the message encryption whereas this part is more complex and thought I would add this instead of clutter

RSA Public Key Encrypting a file:

```
async function encryptFiles(file, publicKeyFile) {
  try {
    // Read binary file as an array buffer
    const fileBuffer = await file.arrayBuffer();

    // Generate a random AES key
    const aesKey = await window.crypto.subtle.generateKey(
      algorithm: {
        name: "AES-CBC",
        length: 256
      },
      extractable: true,
      keyUsages: ["encrypt", "decrypt"]
    );

    // Encrypt the file data using AES
    const iv = window.crypto.getRandomValues(new Uint8Array( length: 16));
    const encryptedFileData = await window.crypto.subtle.encrypt(
      algorithm: {
        name: "AES-CBC",
        iv: iv
      },
      aesKey,
```

```
    fileBuffer                                                    ⚠ 4 ⚠ 17 ✓ 2
  );

  // Export the AES key as a JWK
  const aesKeyJwk = await window.crypto.subtle.exportKey(
    format: "jwk",
    aesKey
  );

  // Read public key file as a string
  const publicKeyString = await publicKeyFile.text();

  // Parse public key from string
  const publicKeyData = publicKeyString.match(/-----BEGIN PUBLIC KEY-----\n([A-Za-z0-9+/\n=]*)\n-----END PU
  const publicKeyBuffer = Uint8Array.from(atob(publicKeyData), c => c.charCodeAt(0));

  // Import public key
  const publicKey = await window.crypto.subtle.importKey(
    format: "spki",
    publicKeyBuffer,
    algorithm: {
      name: "RSA-OAEP",
      hash: {name: "SHA-256"}
```

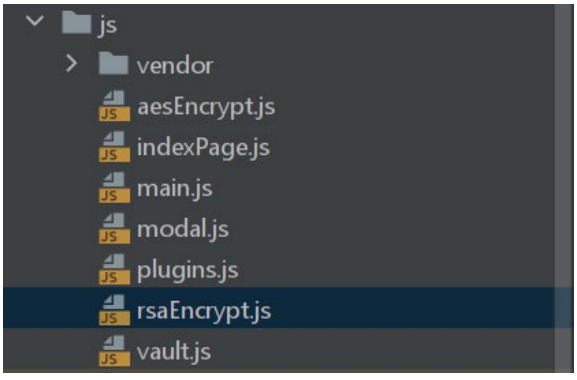
```
// Encrypt the AES key using RSA
const encryptedAesKey = await window.crypto.subtle.encrypt(
    algorithm: {
      name: "RSA-OAEP"
    },
    publicKey,
    new TextEncoder().encode(JSON.stringify(aesKeyJwk))
);

// Concatenate the encrypted AES key and the IV with the encrypted file data
const encryptedFileDataWithHeader = new Uint8Array( elements: encryptedAesKey.byteLength + iv.byteLength + e
encryptedFileDataWithHeader.set(new Uint8Array(encryptedAesKey), 0);
encryptedFileDataWithHeader.set(iv, encryptedAesKey.byteLength);
encryptedFileDataWithHeader.set(new Uint8Array(encryptedFileData), encryptedAesKey.byteLength + iv.byteLe

// Create encrypted file and download link
const encryptedFile = new File([encryptedFileDataWithHeader], file.name + ".enc", {type: file.type});
const encryptedFileUrl = URL.createObjectURL(encryptedFile);
const encryptedFileLink = document.createElement( tagName: 'a');
encryptedFileLink.setAttribute( qualifiedName: "href", encryptedFileUrl);
encryptedFileLink.setAttribute( qualifiedName: "download", value: file.name + ".enc");
encryptedFileLink.click();
URL.revokeObjectURL(encryptedFileUrl);
```

```
  } catch (e) {
    alert("Error encrypting file:\nPlease make sure you have selected a public key file and a file to aesEnc
  }
}
```

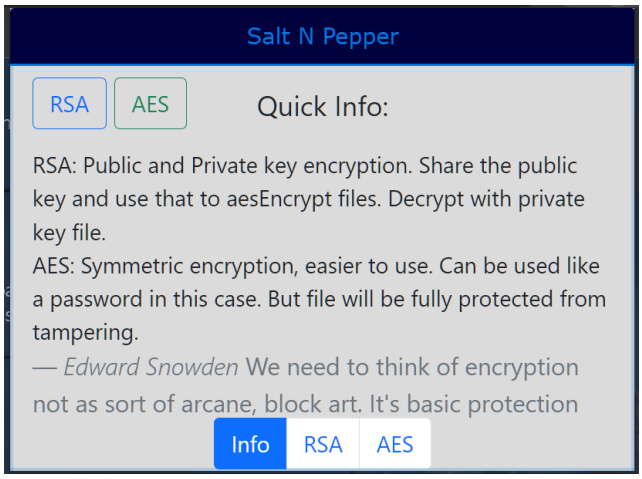
This function is using a hybrid of AES and RSA. As mentioned before RSA is inefficient at encrypting large amounts of data. A workaround for this is to use a hybrid encryption encrypting the file with AES and randomly generating a key based on the key pair. Then that key that is generated is then encrypted with the RSA encryption and key pair, allowing the user to have 2 levels of protection and also making the encryption process efficient.



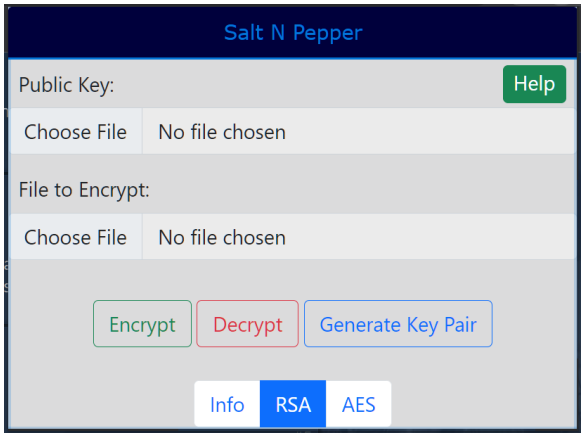
A quick overview of the structure of the functionality of the application. Each functionality of the application is abstracted into its own file and class. This makes it easier to manage and easier to test.

All of the main functions for the application can be viewed here, https://github.com/nathoandrews1/ComputingProject/tree/development/js

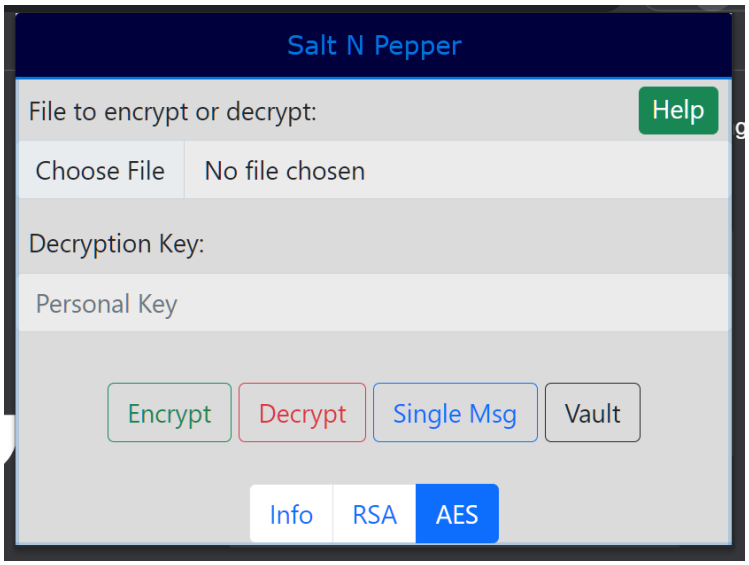
2.4. Graphical User Interface (GUI)



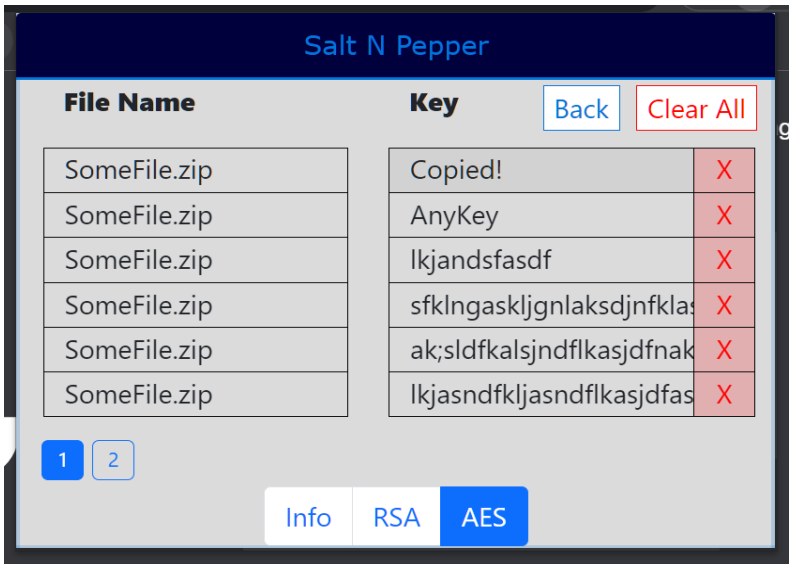
Front page UI, the whole application is dynamic and tries to keep as little information as needed on the screen at a time. Here we can see by clicking the buttons about RSA and AES will give a brief description of the encryption functions for the users to help understand.



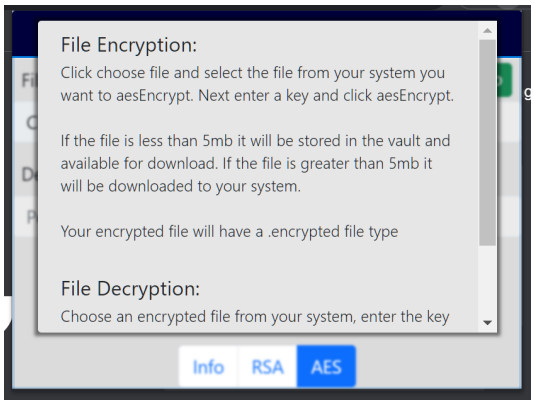
RSA Tab as seen here is for the public and private key encryption. On this page, you will have to generate a key pair which will be downloaded for the user. It is up to the user to keep their private key safe.



AES Encrypt tab, this tab is where the user can use AEs encryption and enter a password to encrypt a file. If the file is less than 5mb it will be saved to the vault where the user can then download the file and view there key associated with that file anytime. There is also another help button to guide the user on how to use it.



Vault tab, this is accessed from the AES tab. This is where the files will be saved if less than 5 MB. When clicking a key it will copy the key to the user's clipboard ready for pasting. Clicking the file name box will trigger a download for the file. There is also a page navigator at the bottom of the page and each page displays a total of 6 results per page.

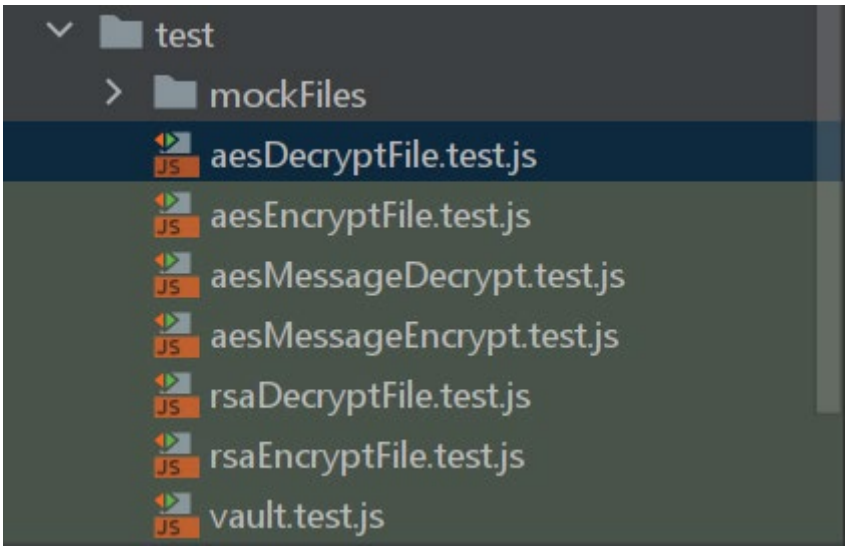


This is a quick example of the Help button for the AES page, this is a popup where the user will scroll to view instructions on how to use it.

2.5.Testing

For testing of the application I have gone with Selenium and Mocha. Selenium will run a browser with the extension loaded simulating a user interacting with the application. This way we can write tests for the functionality of the program to make sure everything is working as it is expected.

The testing development of the application was separated into its own branch and any functionality that is missing tests can be allocated in there with a TODO annotation. For every function that is written, there should be tests written for its functionality. Below are some pictures to better elaborate on how testing has been carried out in the application.



This is an example of the test folder structure. Here we name the test file with the .test in the file name. Any files needed should be added to the mockFiles folder, for example, if we are encrypting and decrypting a file we upload a file here and reference it.

For each test, we will use the describe(“”) syntax that is given with mocha.

We then set up a beforeEach function to setup up some prerequisites.

```
describe( title: 'AES File Decryption', fn: function () {
  this.timeout( ms: 0);
  let driver;

  beforeEach( fn: async function () {
    driver = await new Builder()
      .forBrowser( name: 'chrome') !Builder
      .setChromeOptions(chromeOptions) !Builder
      .build();

    await driver.manage().window().setRect({ width: 375, height: 500 }); // Set the window size
    await driver.get('chrome://extensions');
    const extensionId = 'ipadndgqfcndjdbjiaadoloemlmcnlcb';
    await driver.get(`chrome-extension://${extensionId}/index.html`)
    clearDownloads();
  });

  afterEach( fn: async function () {
    await driver.quit();
  });
```

```
it( title: 'Decrypt a file with AES and check it was downloaded', fn: async function () {
  //Setting up inputs
  const key = '1234567890123456';
  const file = path.resolve( paths: __dirname + '/mockFiles/', 'test.txt');
  const downloadDir ="C:\\Users\\losma\\Downloads";

  //Controlling the encryption flow
  await driver.findElement(By.name( name: "encrypt")).click();
  await driver.findElement(By.id( id: "file")).sendKeys(file);
  await driver.findElement(By.id( id: "key")).sendKeys(key);
  await driver.findElement(By.id( id: "encryptBtn")).click();
  //Wait for file to encrypt
  await driver.sleep(1000);
  await driver.switchTo().alert().accept();
  await driver.findElement(By.id( id: "vaultBtn")).click();
  await driver.findElement(By.name( name: 'fileElement')).click();
  //Wait for file to download
  await driver.sleep(1000);

  // Filter files based on the search string
  let found = false;
  let foundAmount = 0;
  let encryptedFoundAmount = 0;
```

```
  // Filter files based on the search string
  let found = false;
  let foundAmount = 0;
  let encryptedFoundAmount = 0;
  let encryptedFile;
  const filesInDirectory = fs.readdirSync(downloadDir);

  //Check for encrypted file and grab last one in directory
  for(let i = 0; i < filesInDirectory.length; i++){
    if(filesInDirectory[i].toLocaleLowerCase().includes('txt.encrypted')){
      encryptedFoundAmount++;
      encryptedFile = filesInDirectory[i];
    }
  }


  //Controlling the decryption flow
  await driver.findElement(By.name( name: "encrypt")).click();
  await driver.findElement(By.id( id: "key")).clear();
  await driver.findElement(By.id( id: "key")).sendKeys(key);
  await driver.findElement(By.id( id: "file")).sendKeys(downloadDir + "\\" + encryptedFile);
  await driver.findElement(By.id( id: "decryptBtn")).click();
```

```
//Controlling the decryption flow
await driver.findElement(By.name( name: "encrypt")).click();
await driver.findElement(By.id( id: "key")).clear();
await driver.findElement(By.id( id: "key")).sendKeys(key);
await driver.findElement(By.id( id: "file")).sendKeys(downloadDir + "\\" + encryptedFile);
await driver.findElement(By.id( id: "decryptBtn")).click();
//Wait for file to decrypt
await driver.sleep(1000);

// Filter files based on the search string
for(let i = 0; i < filesInDirectory.length; i++){
  if(filesInDirectory[i].toLocaleLowerCase().includes('test.txt')){
    found = true;
    foundAmount++;
    encryptedFile = filesInDirectory[i];
  }
}
if(encryptedFoundAmount > 1){
  foundAmount = foundAmount - (encryptedFoundAmount-1);
}
expect(found).to.equal( value: true);
expect(foundAmount).to.greaterThanOrEqual( value: 1);
```

An example of the test being carried out and at the end of the test we can use the expect() or sometimes known as assert(). We can test the values that are expected of the variables we are testing for. This will then give back results pass or fail. This is to ensure that no regression of the application is happening with any newly added features.

```
DevTools listening on ws://127.0.0.1:49948/devtools/browser/f6ee308c-772d-4334-a787-f4fbe4ce221b
    √ Decrypt a file with RSA (2583ms)


  RSA File Encryption

DevTools listening on ws://127.0.0.1:50060/devtools/browser/0cf747a4-1379-4940-bbf1-d184bcb7356e
    √ Encrypt a file with RSA generate key pair and encrypt file with public key (2459ms)


  Vault Full Test

DevTools listening on ws://127.0.0.1:50110/devtools/browser/8eaaf043-b8c3-4910-a6f2-691a399f27c5
    √ Check All Vault Functionality (21530ms)



  7 passing (56s)

PS C:\____College Year 4\Final_Project>
```

Finally, these are the results of running the test framework and we can see in real-time the application being tested because of selenium.

All of the tests are quite long and can be complex. They all follow the same basic principles we see here and if you would like to see more of the tests please visit the GitHub test branch https://github.com/nathoandrews1/ComputingProject/tree/test_dev/test

2.6. Evaluation

Functionality:
Salt N Pepper offers robust encryption capabilities enabling users to securely encrypt files and text using both RSA and AES encryption methods. These methods have been thoroughly tested and found to perform reliably while ensuring that user data is consistently and effectively protected.

Performance: The performance in particular for handling large amounts of data is highly efficient. This is due in large to the innovative use of hybrid encryption. The extension leverages the speed and efficiency of AES for handling large data, then the resulting AES encryption key is securely encrypted with RSA. This approach allows Salt N Pepper to manage and encrypt large data tasks efficiently without compromising security.

Reliability: The reliability of Salt N Pepper is demonstrated in its robust and resilient design. It consistently performs its intended functions without crashes or errors. All of the areas of the application are covered in tests, providing a dependable solution to users needing secure encryption capabilities. This is further enforced by the use of high-level encryption standards that have been tried and truly tested and are widely known for their reliability and strength in the industry.

3.0 Conclusions

Advantages:

AES Encryption is the highest level of encryption that we have today. Using 256-bit encryption is overkill. 128-bit encryption AES has still not ever been cracked until this day. So, we can guarantee that 256 won't be cracked soon.

AES encryption can serve the full range of file types I am trying to cover for example images. I will have to encrypt the actual pixels of that image and the RGB value. Then the decryption of that image will be reverted back to the original and viewable.

CryptoJS using JavaScript has given the advantage of using CryptoJS Encryption Framework which has been tried and tested in the community.

RSA Encryption: As mentioned before an innovative technique was used in creating the RSA encryption to take advantage of the efficiency of AES for encrypting large pieces of data. This allows a high level of encryption that is unmatched. The application also gives the user the power to store the public and private keys wherever they like. They could even rent their own server and have a pool of friends with access that they can share amongst. It might even be viable in smaller-scale companies with less technical backgrounds.

Disadvantages:

Proposing the application as a Chrome extension may have come as a bit of a disadvantage. Chrome has a problem with persistent data and it is hard to work with rather than being able to store a local JSON file and call values from it, that is not possible. There are many security standards in place when building a Chrome extension for example you can see google has released full documentation on the policy. Content Security Policy: https://developer.chrome.com/docs/apps/contentSecurityPolicy/ This will pose a challenge when implementing the store keys section.

Chrome extension again does not give access to the user's local files or directory. Making the vault a hard functionality to implement. This means having to use the chrome storage API. This is not exactly ideal and would be better to go against what I had thought initially and host a server with a database for this purpose only. Which would allow users to have greater and also more secure options. This could possibly be implemented as a paid service of the app.

Chrome Extension building also requires that you cannot outsource your stylesheets or scripts. All files must be packaged with the extension. This comes as a disadvantage to production time and requires a bit more manual implementation.

4.0 Further Development or Research

With further development I would like to add for the users to be able to store data items to a vault larger than 5 MB. This would require having a server and hosting a database. If the app gained any traction it could possibly be implemented as a paid service. This would really help with the running costs.

Research into encrypting and decrypting with QR-type passwords where we can generate a QR code that can be used to encrypt data. This would allow for a mobile type of digital key possibly allowing for new types of digital locks.

5.0 References

6.0 swampsjohnswampsjohn 6 *et al.* (1957) *How to test chrome extensions?*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/2869827/how-to-test-chrome-extensions (Accessed: December 17, 2022).

6.0 Appendices

6.1. Project Proposal

_X19135394_Project Proposal_NathanA (1)

6.2. Reflective Journals


x19135394_Month1_
ReflectiveJournal.docx


x19135394_Month2_
ReflectiveJrounal.docx


x19135394_Month3_
ReflectiveJrounal.docx


x19135394_Month4_
Jan_ReflectJournal.do


x19135394_Month5_
Feb_ReflectJournal.do


x19135394_Month6_
Marc_ReflectJournal.d


x19135394_Month8_
April_ReflectJournal.d