

ENHANCED USER DATA STORAGE WITH CRYPTOGRAPHY

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

By

SRIRAM.S (Reg. No. 39290105)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119, TAMILNADU

MARCH 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to verify that this Project Report is the bonafide work of **SRIRAM . S (Reg. No. 39290105)** who carried out the project entitled as "**ENHANCED USER DATA STORAGE WITH CRYPTOGRAPY**" under my supervision from . Dec 2020 to march 2022

Internal Guide

Dr.Mohanprasad ,M.E.,Ph.d.,

Head of the Department

DR. L .Lakshmanan M.E, Ph.D

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **SRIRAM.S (Reg. No. 39290105)** hereby declare that the Project Report entitled "ENHANCED USER DATA STORAGE WITH CRYPTOGRAPHYSs" done by me under the guidance of **Dr.Mohanprasad ,M.E.,Ph.d.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Science degree in Computer Science.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D.**, Dean, School of Computing and **Dr. S. VIGNESHWARI, M.E., Ph.D.**, and **Dr. L. LAKSHMANAN, M.E., Ph.D.**, Head of the Department, Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr. mohanradhakrishnan ., for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my projectwork.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of Computer Science and Engineering who were helpful in many ways For the Completion of the project.

ABSTRACT

The Data security is one of the most important topics in an Internet first world.

Millions of user records are stored in here by various web and mobile applications every day and they very easily get hacked or leaked by malicious user very often due to lack of security which is a very critical level problem today in an internet first world. Also we should make sure that it is the need of the hour to ensure data security and user privacy.

In our approach, we proposed a novel idea to enable user to securely store their data through special techniques such as encryption and decryption of data into 2048 bits and etc. technique and we mainly improve awareness about such topics at the same time.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	vii
		i
1	INTRODUCTION	1
	1.1 OVERVIEW OF PROJECT	1
2	LITERATURE SURVEY	2
3	AIM AND SCOPE OF PRESENT INVESTIGATION	4
	3.1 AIM OF THE PROJECT	4
	3.2 SCOPE AND OBJECTIVE	4
	3.3 SYSTEM REQUIREMENTS	6
	3.3.1 HARDWARE REQUIREMENTS	6
	3.3.2 SOFTWARE REQUIREMENTS	6
	3.4 SOFTWARE USED	7
	3.4.1 PYTHON BACKEND	7
	3.4.2 FAST API FRAME WORKS	7
	3.4.3 JAVA SCRIPT LANGUAGE	7
	3.4.4 VUE JS FRAME WORK	8
	3.4.5 VS CODE	8

4	METHODOLOGY	10
	4.1 RSA ALGORYTHM	10
	4.1.1 EXISTING SYSTEM	10
	4.1.2 PROPOSED SYSTEM	10
	4.2 MODULE DESCRIPTION	11
	4.3 FRONT END DISCRPTION	16
5	RESULTS AND PERFORMANCE ANALYSIS	20
	5.1 RESULT	20
	5.2 PERFORMANCE ANASLYSIS	21
6	CONCLUSION AND FUTURE ENHANCEMENT	22
	6.1 CONCLUSION	22
	6.2 FUTURE ENHANCEMENT	22
	REFERENCES	23
	APPENDIX	24
	A. SOURCE CODE	24

LIST OF FIGURES

FIGURE NO:	FIGURE NAME	PAGE NO
4.1	RSA algorithm	10
4.2	User management in web application	11
4.3	Type of User management	11
4.3	Key management In web application	12
4.4	Types of key management	12
4.5	Crypto management in web application	13
4.6	Type of crypto management	13
4.7	Architecture diagram	14
4.8	User register page	16
4.9	user login page	17
4.10	user after login page	17
4.11	user data entering	18
4.12	user data encryption	18
4.13	user data decryption	19

CHAPTER 1

1. INTRODUCTION

1.1 OVERVIEW OF PROJECT

What is data security?

Data security is the practice of protecting digital information from unauthorized access, corruption, or theft throughout its entire lifecycle.

DATA Security means protecting digital data , such as those in a database , from destructive forces and from the unwanted actions of unauthorized users , such as cyber attack or a data breach. Web application security is so important In these days that we can prevent losing of any sensitive data through cyberattack or hacking and data breach .

DATA SECURITY is the most important topic in our day today internet first world.

The Web application security is very crucial to protecting data for customers and organization from data pilferage and other harmful results of cyber crime.

In such complications, we make it simple and secure by encrypting and decrypting the data, with a unique key per user. .

CHAPTER 2

2. LITERATURE SURVEY

1. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

Authors: R.L. Rivest, A. Shamir, and L. Adleman

RSA (Rivest–Shamir–Adleman) is a widely used [public-key cryptosystem](#) used to secure data. The [acronym](#) "RSA" comes from the surnames of [Ron Rivest](#), [Adi Shamir](#) and [Leonard Adleman](#), who invented the algorithm. An equivalent system was developed by English mathematician [Clifford Cocks](#).

Public-key [cryptosystem](#) consist of two non identical keys. The [encryption key](#) is public and distinct from the [decryption key](#), which is kept secret (private). An RSA user creates and publishes a public key based on two large [prime numbers](#), along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers.

The length of the key forms a critical part in determining the security of the algorithm. There are various key length such as 256, 512, 1024, 2048, bits. As computational power increases key length is also increased proportionally.

2. OpenPGP Message Format Callas, J., Donnerhacke, L., Finney, H., and R. Thayer

Pretty Good Privacy (PGP) is an [encryption program](#) that provides [cryptographic privacy](#) and [authentication](#) for [data communication](#). PGP is used commonly in encrypting, and decrypting texts, signing, files, emails, directories, and to increase the [security](#) of e-mail communications. [Phil Zimmermann](#) invented PGP in 1991.

[OpenPGP](#) is an open standard of PGP encryption [software](#), standard (RFC 4880) for encrypting and decrypting [data](#).

PGP encryption uses a combination of techniques such as , , symmetric-key cryptography, data compression, hashing, public-key cryptography; each step uses one of several supported algorithms. Each public key is bound to a username or an e-mail address. The first version of PGP encryption was generally known as a web of trust to contrast with the X.509 system, which uses a different approach based on certificate authority and which was added to PGP implementations later. Current versions of PGP encryption include options through an automated key management server.

3. Data Encryption and Decryption Using RSA Algorithm in a Network Environment". Nentawe Y. Goshwe Arham Chopra ,“

Security has become a wide necessity in everyday life. The most important security of all is data security. Data is exposed to high potential risks by various platform in our system. The authors have adopted various methods for various security reasons. Now everyone is dependent on the security and storage cloud platform, but cloud environment is as well vulnerable to different threats. The cloud data is not good-secured as it can be accessed by anyone who can reach our credentials, and cloud providers also have equal access to us. Thus, using encryption and steganography, the authors have propose enhanced data security. Here the data is encrypted and hidden behind an image and subsequently uploaded to the cloud. The image could be downloaded whenever it felt necessary and the data can be decrypted to retrieve the original file. The results of this paper provide improved data security and can be used smoothly anywhere.

CHAPTER 3

3. AIM AND SCOPE OF PRESENT INVESTIGATION

3.1 AIM OF THE PROJECT

Genre classification is an important task with many real world applications. to develop a deep learning project to automatically classify different musical genres from audio files. We will classify these audio files using their low-level features of frequency and time domain. we need a dataset of audio tracks having similar size and similar frequency range. GTZAN genre classification dataset is the most recommended dataset for the music genre classification

3.2 SCOPE AND OBJECTIVE

Providing a secure data storage option to user.

User's data has been stored in a very very secure way where decoding the encrypted message is an impossible way.

User can upload the files to the web portal.

user can upload their sensitive data through web portal itself which is secured by private and public key.

Ensure only owner of the data can access the information even incase of cyber attack.

Even in a cyber attack the data which has been stolen by the hackers can decoded by the user with the key given to the user.

Utilize efficient cryptographic algorithms like RSA, elliptic curve algorithms to encrypt data fast and real time.

Very efficient cryptographic algorithms like RSA, Elliptic curve have been used as it is very fast and easy to crypt and decrypt.

One secure key per user during profile creation.

Every user has been given an unique secure key while creating the user account in the web server.

All user keys are encrypted with master key

every single user key is secured and protected with a single master key which is with the webserver.

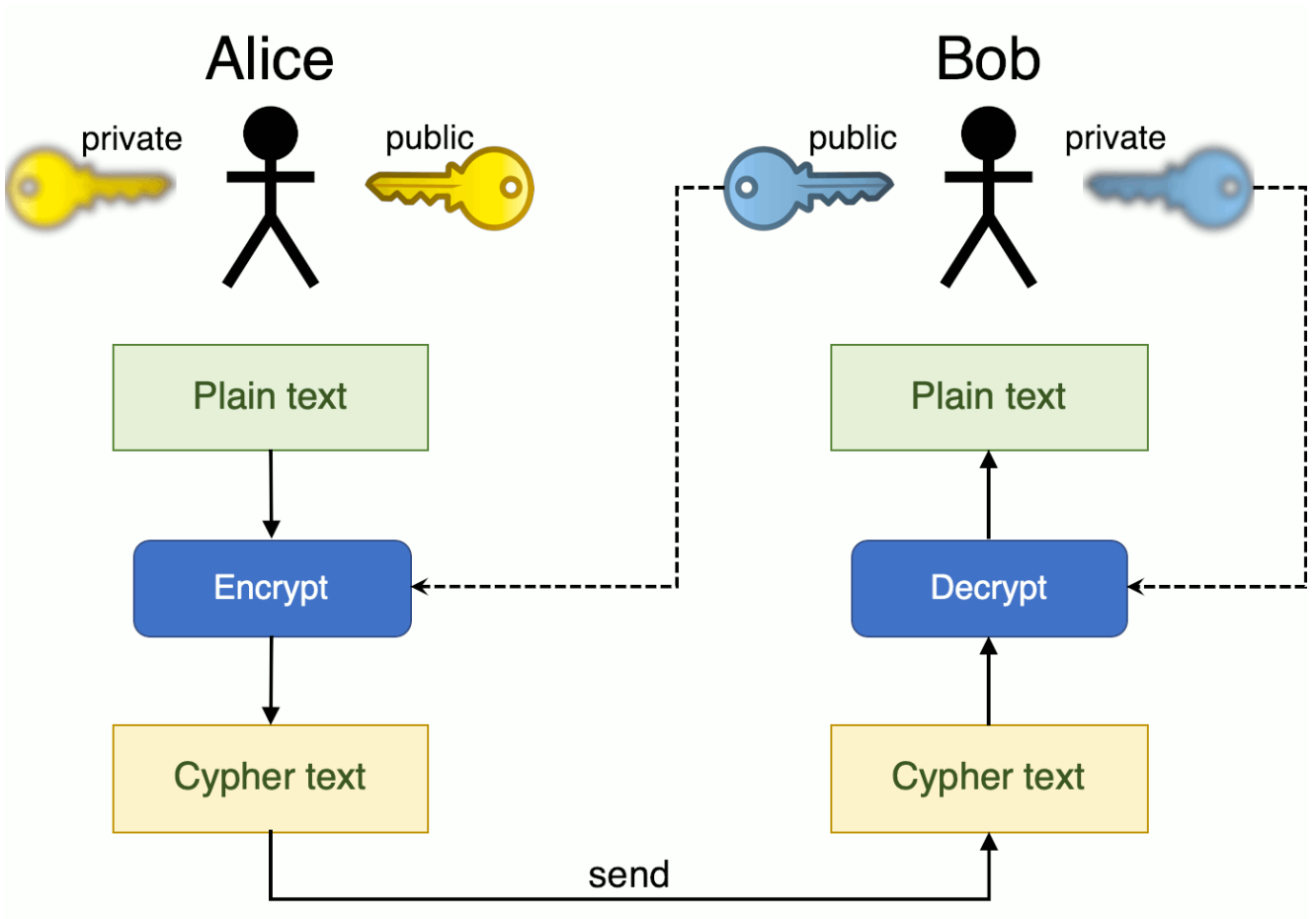


Fig 3.1

3.3 SYSTEM REQUIREMENTS

3.3.1 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. The minimal hardware requirements are as follows,

Server side:

- CPU 2GB ram
- Hard-disk: 50GB
- Python 3.9 or higher
- Postgres 13.6 or higher

Client Side:

- CPU: 4GB
- Operating system: any
- Latest browser version of any browser

3.3.2 Software Requirements

Software requirements deals with defining resource requirements and prerequisites that needs to be installed on a computer to provide functioning of an application. The minimal software requirements are as follows,

1. Front end : Java script
2. Back end :Python
3. IDE : Vs code
4. Operating System : Windows

3.4 SOFTWARE USED:

3.4.1 Python Language

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language include NASA, Google, YouTube, BitTorrent, etc. Python programming is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python had deep focus on code readability & this class will teach you python from basics.

3.4.2 FAST API

- In python, fast api is a webframework which is used for developing RESTful APIs .
- It can fully support asynchronous programming
- It can easily set up and run with Uvicorn and Gunicorn.
- From the earliest days of the project, editor support was considered to improve developer-friendliness of the project.
- It is far more lighter than Django and offers more similar features to flask.
- It is built with async in mind.

3.4.3 JAVA SCRIPT LANGUAGE:

JAVA script tool is very important tool for front end developer.

It has a powerful framework tool which can help to render a page. These are also used in typical situations with complex dynamics interactions that needed to occur.

We can resolve many complex issues using java script framework and complete your expected form and make your clients requirement fulfill.

Without java script webpages could not have become dynamic web application. No image carousels would be there without java script. without java script there could not be a partial page which when reloads , keeps your spot on the current page.

There would not be many things which are the things we are currently, accustomed in our day today life .

3.4.4 Vue js framework

For building user interfaces Vue is a very progressive framework.

It is incrementally adoptable as it is designed ground up.

Since the core library is focused on the view layer,

it is very easy an simple to pickup and integrate with other libraries or existing projects.

Very sophisticated single –page is perfectly and efficiently handled by vue.

Also handles applications used in combination with modern tooling and supporting libraries.

3.4.5 VS Code:

VS Code is a free, open source streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more. Additionally, the Python Extension for Visual Studio Code tailors VS Code into a Python IDE. It is a streamlined code editor with support for development operations like debugging, task running and version control.

VS Code is free for both private and commercial use, runs on Windows, macOS, and Linux, and includes support for linting, debugging, task running, version control and Git integration, IntelliSense code completion, and conda environments. VS Code is openly extensible and many extensions are available.

Visual Studio Code is a streamlined code editor with support for development operations like debugging, task running, and version control. It aims to provide just the tools a developer needs for a quick code-build-debug cycle and leaves more complex workflows to fuller featured IDEs, such as Visual Studio IDE.

4. EXPERIMENTAL OR MATERIAL METHODS

4.1 RSA algorithm

RSA algorithm is a asymmetric algorithm. Which actually means that RSA algorithm works on two different keys , they are public and private key. Public key is given to everyone and the private key is stored in private.

For example , a browser sends its public key and asks for some data. Then the server sends the encrypted data to the server which is encrypted using server's public key, Then the server recives it and decrypts the data.

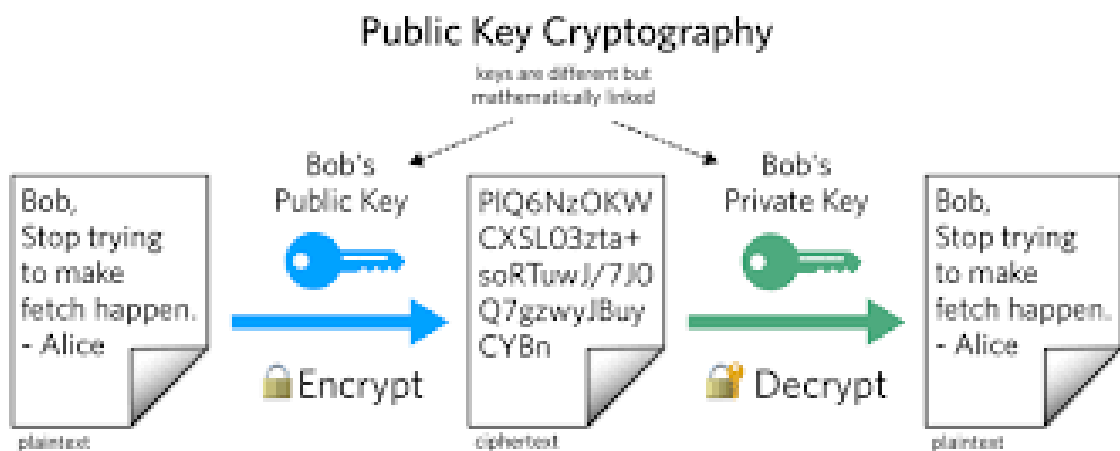


Fig 4.1

4.1.1 Existing System:

512-bit key is forbidden key as it can be easily cracked. In 1999 August 512-bit has been factorized as the challenge for breaking this key code was successfully completed nowadays it would be a lot easier than those days to crack such combinations.

The current record is 768.

4.1.2 Proposed System:

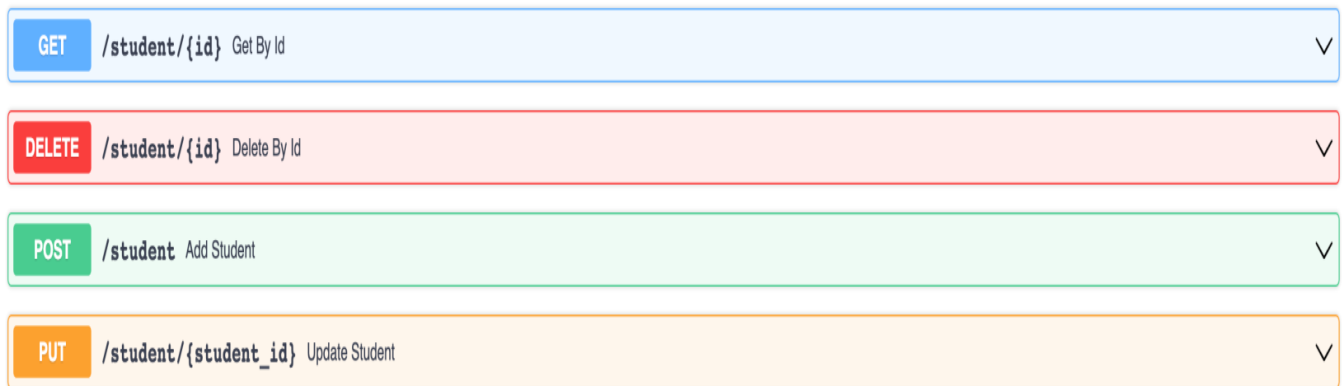
The end user can very easily download and upload their required data in a very safe and secure manner. We make sure that the actual owner of the specific data can only access it. We have stored the data safely through cryptographic encryption. The data is encrypted when the user uploads it with his private key . using the private key when the user downloads the data it is decrypted.

4.2 MODULE DISCRIPTION

user management

Through these functions we can easily handle such functions as read,update and delete of the user data . all the related functions and calls of these functions are in backend.

The 'USER' in the database table is used for the relevant process.



The image shows a list of four REST API endpoints for user management, each in a colored box with a dropdown arrow on the right:

- GET** /student/{id} Get By Id (light blue box)
- DELETE** /student/{id} Delete By Id (light red box)
- POST** /student Add Student (light green box)
- PUT** /student/{student_id} Update Student (light orange box)

FIG 4.2

Columns



		Name	Data type
		id	bigint v
		first_name	text v
		last_name	text v
		email	text v
		password	text v

FIG 4.3

Module 2: key management module

Through these functions we can easily handle create, read of the secure key data. All the related API calls are in backend. The 'user-key' in the database table is used for relevant process.

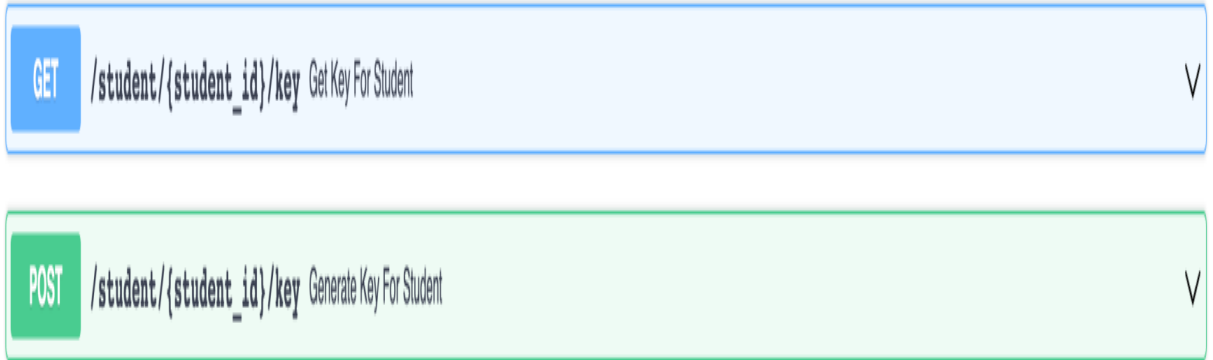


FIG 4.3

Columns













		Name	Data type
		id	bigint 
		public_key	text 
		private_key	text 
		student_id	integer 

FIG 4.4

Module 3: crypto management module

We can handle encryption and decryption of the user data. All the related calls are in the backend. Here we get data from the user, encrypt them and store safely in the db. Then reads the cipher texts from the db, decrypts the data and sends to the front end. 'user-key' is used for the relevant process of the data base.

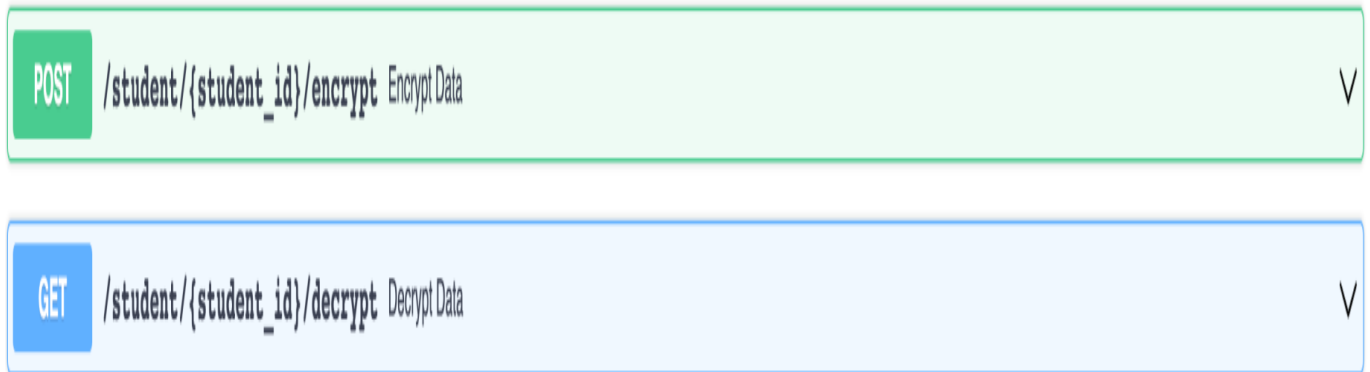


FIG 4.5

Columns

		Name	Data type
		id	bigint v
		student_id	integer v
		hint	text v
		encrypted_message	text v

FIG 4.6

Architecture diagram

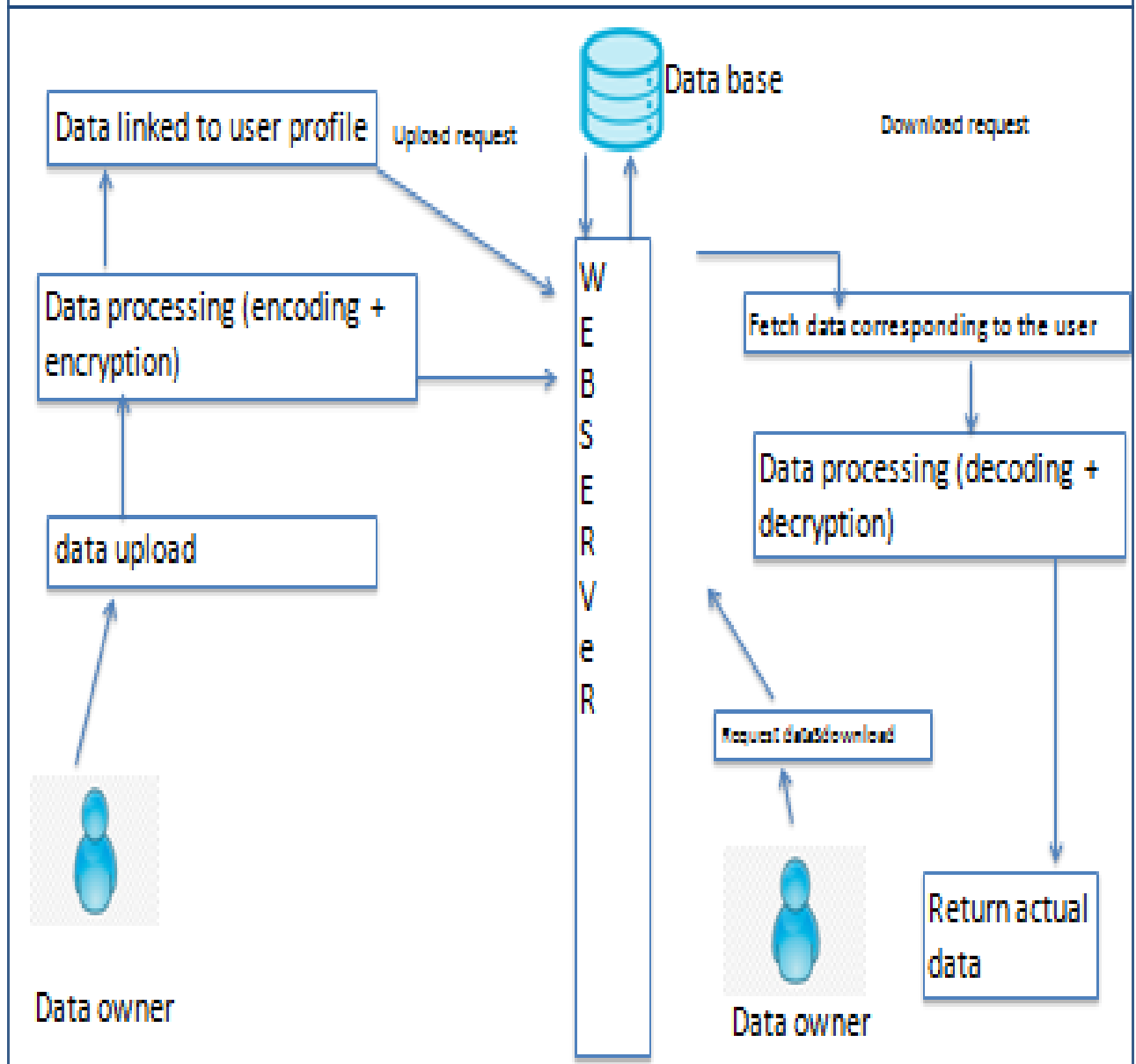


Fig 4.7: Architecture Diagram

ARCHITECTURE DIAGRAM EXPLANATION.

The Data owner will upload the data to the server, then the data which is uploaded by the user to the server is data processed and encoded with encryption. Then the data is linked to the user with a unique key which is given separately for every data owner separately. Then Finally the encrypted data is uploaded to the web server safely. This is how the owner's data has been secured safely. When the data owner requests the uploaded file to the database then the downloading request processes takes place. Where the Data will be fetched for the corresponding user from the web server which is then verified with the user key and then decoding process takes place. Then downloaded data is decoded and decrypted. Finally the requested data by the user is returned to the data owner.

4.3 Front end description

User registration / login page

[Home](#) | [Add New Message](#) Log-in

Data Locker

Your data is secure with us!!!

User Registration

Enter your first name:

Enter your last name:

Enter your email:

Enter your password:

Retype your password:

Register

FIG 4.8

This is the screenshot of the login page where the user logs in using his user id and password.

Before login:

[Home](#) | [Add New Message](#) [Log-in](#)

Data Locker

Your data is secure with us!!!

Enter your email:

Enter your password:

[Sign-in](#)

Do not have an account yet? [Register here](#)

FIG 4.9

After login:

[Home](#) | [Add New Message](#) [Your Profile](#)

Data Locker

Your data is secure with us!!!

Label	Value
First Name	Ankith
Last Name	Shah
Email	ankiths@gmail.com

[Sign-out](#)

Fig 4.10

Login page for already existing user, where the user can login to access his data by using his email and his secure password.

User Entering the data with hint for easy reference.

[Home](#) | [Add New Message](#) Your Profile

Data Locker

Your data is secure with us!!!

Add a new message

Enter your message:

Message

Enter your hint:

Hint

Store Message

Fig 4.11

User's data where every data is secured with special unique key.

[Home](#) | [Add New Message](#) Your Profile

Data Locker

Your data is secure with us!!!

Your stored messages

Hint	Action	Message
secret hint	decrypt	
passport data	decrypt	
My super long message	decrypt	

Fig 4.12

User after entering the secure key and decrypting the data.

[Home](#) | [Add New Message](#) Your Profile

Data Locker

Your data is secure with us!!!

Your stored messages

Hint	Action	Message
secret hint	decrypt	My very secret message
passport data	decrypt	my passport information for government
My super long message	decrypt	Compared to display: inline, the major difference is that display: inline-block allows to set a width and height on the element. Also, with display: inline-block, the top and bottom margins/paddings are respected, but with display: inline they are not. Compared to display: block, the major difference is that display: inline-block does not add a line-break after the element, so the element can sit next to other elements. The following example shows the different behavior of display: inline, display: inline-block and display: block

Fig 4.13

CHAPTER 5

5. RESULTS AND PERFORMANCE ANALYSIS

5.1 RESULT

RSA key used in our system

As 512-bit keys are not providing sufficient security and should be minimized in secure / confidential projects or for usage in security because of lack of rigid security. So we have used 2048 bits for extreme security for valuables like data and etc.

The 2048 key is 4 times efficient than 512 and 2 times efficient than 1024 bits.

Our approach is more durable and efficient in securing user data.

Our algorithm users 2048 bit key length to secure the data, which is the industry standard.

5.2 PERFORMANCE ANALYSIS

Key length	Security effectiveness
128	10
256	25
512	40
1024	60
2048	95

Table 5.1

CHAPTER 6

6. CONCLUSION AND FUTURE ENHANCEMENT

6.1 CONCLUSION:

We would like to address the multi-class solution to problems such as data theft with an ensemble of Machine Learning methods that could prevent loss of many sensitive data, with using RSA and CRYPTORAPHY methods. This would encrypt the user's data which is then encoded with symmetric encryption with separate public and private keys. We not only provide only security but also sureity for safe data which is very very hard for an outsider to decrypt it.as we have used 2048 RSA-CRYPTOGRAPHY encryption.as 512 and 1024 key codes can be easily de coded using today's morden computer and super computer. As 2048 key code is 2 times more secure than 1024 and 4 times more secure than 512 key code .

6.2 FUTURE ENHANCEMENTS:

In the future, we hope to experiment with other types of deep learning methods, given that 2048 key codes can be decrypted using advanced super computers. As for now 2048 performed the best. Even if 2048 is decrypted in future using super computers we may also have improved key codes which will be harder for future computers which will be enhanced by the future generations.

REFERENCES

- [1] R.L. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", [RFC 1421](#), February 1993.
- [2] Callas, J., Donnerhackle, L., Finney, H., and R. Thayer, "OpenPGP Message Format", [RFC 2440](#), November 1998.
- [3] Nentawe Y. Goshwe Arham Chopra ,“Data Encryption and Decryption Using RSA Algorithm in a Network Environment”.
- [4] Schönbrodt FD, Humberg S (2021). RSA: An R package for response surface analysis (version 0.10.4).
- [5] <https://cran.r-project.org/package=RSA>.
- [6] <https://datatracker.ietf.org/doc/html/rfc4492>
- [7] <https://core.ac.uk/download/pdf/231151959.pdf>

APPENDIX

A. SOURCE CODE

FRONT END SOURCE CODE:

For user login

```
<template>
  <div v-if="loginFailure" class="alert alert-danger AlertBanner" role="alert">
    Login failed
  </div>

  <div>
    <div v-if="this.$store.state.loggedIn">
      <div>
        <table>
          <tr>
            <th>Label</th>
            <th>Value</th>
          </tr>
          <tr>
            <td>First Name</td>
            <td>{{this.firstName}}</td>
          </tr>
          <tr>
            <td>Last Name</td>
            <td>{{this.lastName}}</td>
          </tr>
          <tr>
            <td>Email</td>
            <td>{{this.email}}</td>
          </tr>
        </table>
      </div>
      <button class="SignOut Button" @click="logout"> Sign-out </button>
    </div >
    <div v-else-if="this.$store.state.loggedIn == false && registerFirstTime == false">
      <div class="LoginForm">
        <label class="Label">Enter your email:</label><br>
        <input class="UserName" type="text" v-model="email" placeholder="E-mail"
      />
        <label class="InputError" v-if="emailError ">Username should not be
empty</label><br>
        <label class="Label">Enter your password:</label><br>
        <input class="password" type="password" v-model="password"
placeholder="Password"/>
        <label class="InputError" v-if="passwordError">Password should not be
empty</label><br>

```

```

    </div>
    <button class="Button" @click="login"> Sign-in </button>
    <p class="Register"> Do not have an account yet? <a href="/registerUser"
    @click="register"> Register here </a> </p>
  </div>
</div>

```

```
</template>
```

```

<script>
import axios from "axios"
import { useCookies } from "vue3-cookies";

```

```

export default {
  name: 'Login',
  setup() {
    const { cookies } = useCookies();
    return { cookies };
  },
  components: {

},
  props: {
}, data(){
  return {
    email: "",
    password:"",
    firstName : "",
    lastName:"",
    registerFirstTime:false,
    emailError:false,
    passwordError:false,
    loginFailure:false,
    loginSuccess:false,
    allMessages:[],
  }
},
  mounted(){
    this.loginSuccess = this.$store.state.loggedIn
    if(this.loginSuccess){
      const info = this.$store.state.userInfo
      console.log(info)
      const infoarray = info.split('*')
      console.log(infoarray)
      this.firstName = infoarray[0]
      this.lastName = infoarray[1]
      this.email = infoarray[2]
    }
  },
  methods:{
    login(){
      if(!this.email){

```

```

    this.emailError= true
  }
  if(!this.password){
    this.passwordError= true
  }
  if(!this.emailError && !this.passwordError){
    axios.post('http://localhost:8000/login', {
      'email':this.email,
      'password':this.password
    })
    .then((response)=>{
      this.cookies.set("studentId", response.data.id);
      this.loginSuccess= true
      this.emitter.emit('Loginstatus', 'finished')
      this.$store.commit('confirmLogin')
      this.$store.commit('setUserInfo',
response.data.firstName+'*'+response.data.lastName+'*'+response.data.email)
      this.$router.push({ path: '/' })
    })
    .catch(error=>{
      this.loginFailure = true;
      console.log(error);
    });
  }
},
logout(){
  this.cookies.remove("studentId");
  this.emitter.emit('Logoutstatus', 'finished')
  this.$store.commit('confirmLogout')
},
register(){
  this.registerFirstTime = true
}
}
}
</script>

```

```

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
ul {
  list-style-type: none;
  padding: 0;
}
li {
  display: inline-block;
  margin: 0 10px;
}
a {
  color: #42b983;
}
header{
  background-color: #42b983;
  padding:8px
}

```

```

.Label{
  width: 100%;
  text-align: left;
  font-weight:bold;
}
.UserName{
  width:100%
}
.password{
  width:100%
}
.LoginForm{
  margin: 10px;
}
.InputError{
  color: red;
  text-align: left;
  width:100%;
  font-weight:bold;
}
table{
  margin:50px;
  margin-left: auto;
  margin-right: auto;
}
th{
  text-align:left;
}
tr{
  width: 150px;
  margin:50px;
  text-align:left;
}
td{
  width: 150px;
  margin:50px;
  text-align:left;
}
.SignOut{
  float: right;
  margin:30px
}
.Button{
  background-color: #42b983;
}
.Register{
  margin:20px
}
</style>

```

For Storing messages:
<template>

```

<div class="Root">
  <div>
    <p class="StoredMsg"> Your stored messages</p>
  </div>
  <div class="Holding">
    <div class="Heading">
      <label class="Hint"> Hint </label>
      <label> Action </label>
      <label class="Message"> Message </label>
    </div>
    <div class="Row" v-for="item in allMessages" :key="item.id">
      <MessageItem :message = "item"></MessageItem>
    </div>
  </div>
</div>
</template>

<script>
import axios from "axios"
import MessageItem from "./MessageItem.vue"
// import Header from "./Header.vue"
import { useCookies } from "vue3-cookies";

export default {
  name: 'HelloWorld',
  setup() {
    const { cookies } = useCookies();
    return { cookies };
  },
  components: {
    MessageItem,
  },
  props: {
    msg: String
  }, data(){
    return {
      logInComplete:false,
      allMessages:[],
    }
  },
  created(){
  },
  mounted(){
    this.logInComplete = this.$store.state.loggedIn
    this.checkLogInComplete()
    this.getData()
  },

  methods:{
    getData(){
      const studentId = this.cookies.get("studentId");
      axios.get(` http://localhost:8000/student/${studentId}/messages`)
        .then((response)=>{
          let messages = response.data.messages;

```

```

        this.allMessages = messages
        this.allMessages = this.allMessages.map(item =>({...item,
studentId:studentId}))
    });
},
checkLogInComplete(){
    if(!this.logInComplete){
        this.$router.push({ path: '/login' })
    }
}
}
}
}
</script>

```

<!-- Add "scoped" attribute to limit CSS to this component only -->

```

<style scoped>
ul {
    list-style-type: none;
    padding: 0;
}
li {
    display: inline-block;
    margin: 0 10px;
}
a {
    color: #42b983;
}
header{
    background-color: #42b983;
    padding:8px
}
.StoredMsg{
    font-weight: bold;
    text-align: center;
    width:100%;
    padding-top: 10px;
}
.Holding{
    margin:10px;
}
.Heading{
    display: flex;
    justify-content: space-between;
}
.Row:nth-child(even) {background-color: #f2f2f2;}

label{
    /* display: inline-block; */
    /* margin-left: 108px; */
}
.Hint{
    margin-left: 16%;
}

```

```

.Message{
  margin-right: 14%;
}
</style>

```

For reading messages:

```

<template>
<div class = "MessageItem">
  <span class="Label"> {{message.hint}}</span>
  <button class="Button" @click="decryptMessage(message.hint,
    message.studentId)"> Decrypt</button>
  <span class="Message"> {{decryptedMessage}}</span>
</div>

</template>

<script>
import axios from "axios"
export default {
  name: 'HelloWorld',
  props: {
    message: {}
  },
  data(){
    return {
      decryptedMessage:"
    }
  },
  methods:{
    decryptMessage(hint, studentId){
      axios.get(` http://localhost:8000/student/${studentId}/decrypt?hint=${hint}`)
      .then((response)=>{
        console.log(response);
        this.decryptedMessage = response.data.decryptedMessage
      }).catch((err)=>{
        console.log(err);
        alert('Message Decryption Failed!!! Contact your Admin')
      });
    }
  }
}
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>

.MessageItem{
  text-align:center;
  width: 100%;
  display: flex;

```

```

        justify-content: space-between;
        margin-top: 10px;
        margin-bottom: 10px;
    }
    .Label{
        width: 33%;
    }
    .Message{
        width: 33%;
    }
    .Button{
        height: 30px;
        background-color: #42b983;
    }
    /* .Label{

        display: inline-block;
        width: 87px;
        margin-top: 7px;
        margin-bottom: 7px;
        text-align: center;
        padding-left: 60px;
    } */
    /*
    .Message{
        padding-left: 120px;
        display: inline-block;
        width: 87px;
        margin-top: 7px;
        margin-bottom: 7px;
        text-align: center;
    }

    .Button{

        display: inline-block;
        margin-left: 100px;
        margin-top: 7px;
        margin-bottom: 7px;
        text-align: center;
        background-color: #42b983;
    } */

</style>

```

Back end source code:

For encryption and decryption:

class MessageController:

```

    def encryptMessage(self, student_id:int, message:str, hint:str):
        student_controller = StudentController()
        student:StudentResponseModel | None =
student_controller.read_student_by_id(student_id)

```



```

database = Database()
student_key = database.read_key_for_student(student.id)
if student_key is not None:
    rsa_service = RsaService()
    encrypted_message = rsa_service.encrypt_data( student_key.private_key,
message)
    stored_data = database.store_encrypted_message(student_id, hint,
encrypted_message)
    return EncryptedMessageResponseModel(id=stored_data.id, hint =
stored_data.hint, encryptedMessage = stored_data.encrypted_message)
else:
    raise NotFoundException(f"Key for student with id {student_id} not found")

def decryptMessage(self, student_id:int, hint:str):
    student_controller = StudentController()
    student_key_controller = StudentKeyController()
    database = Database()

    student:StudentResponseModel | None =
student_controller.read_student_by_id(student_id)
    student_key = database.read_key_for_student(student.id)
    encrypted_message = database.read_encrypted_message(student_id, hint)
    if encrypted_message is not None and student_key is not None:
        rsa_service = RsaService()
        decrypted_message = rsa_service.decrypt_data(
            student_key.private_key,
            encrypted_message.encrypted_message
        )
    elif encrypted_message is None:
        raise NotFoundException(f"Student with id {id} has no message with hint
{hint}")
    elif student_key is None:
        raise NotFoundException(f"Key for student with id {student_id} not found")
    return DecryptedMessageResponseModel(
        id=encrypted_message.id,
        decryptedMessage = decrypted_message,
        hint = hint
    )

def readAllMessageaForStudent(self, student_id:int):
    student_controller = StudentController()
    student:StudentResponseModel | None =
student_controller.read_student_by_id(student_id)
    database = Database()
    messages = database.read_all_encrypted_message_for_student(student_id)
    encryptedMessages = []
    if messages is not None:
        for message in messages:
            encryptedMessages.append(EncryptedMessageResponseModel(id=message.id,
hint = message.hint, encryptedMessage = message.encrypted_message))

```

```
        return  
AllEncryptedMessageResponseModel(messages=encryptedMessages)  
    else:  
        raise NotFoundException(f"Student with id {id} has no encrypted message")
```