



UNIVERSITY OF PETROLEUM AND ENERGY STUDIES, DEHRADUN

Zip Guard

“Shield your data with Encryption”

Final Report of the (Minor Project - 1) in Semester V

Prepared by Team No: 24

S. No	Students Name	Roll Number	Sap Id
1.	Akshat Pandey	R2142220306	500101788
2.	Madhav Madan	R2142220265	500105699
3.	Dhruv Joshi	R2142220483	500106021
4.	Kavya Singh	R2142220538	500106891

BACHELOR OF TECHNOLOGY, COMPUTER SCIENCE ENGINEERING

With specialization in DevOps

Under the guidance of

Dr. Arjun Arora


13/12/24

INDEX

S.No	Heading Outline	Page no.
1	Title of the project	3
2	Introduction	3
3	Background Information	3
4	Motivation	4
5	Related work	5
6	Problem Identification	5
7	Problem statement	6
8	Objective	6
9	Methodology	7
10	Hardware Requirement	8
11	Software Requirement	8
12	Results	9
13	SWOT Analysis	13
14	Conclusion	13
15	References	14
16	Approval by mentor	14

Zip Guard

"Shield your data with Encryption"

Introduction

The protection and management of data are critical in the digital age. "Zip Guard" is a tool with strong file compression and encryption features that is made to meet these objectives. This project allows users to secure their data via encryption and decryption, as well as zip and unzip files using Huffman coding. With the added convenience of auto-generated keys, users have a choice between two robust encryption algorithms: AES and RSA. Zip Guard's main goal is to provide a straightforward but efficient method of safeguarding confidential information.

Background Information

Encryption and file compression are two essential methods for today's data management. A popular technique for efficient data compression that minimizes file size without sacrificing integrity is Huffman coding. However, encryption makes sure that the data is compressed and protected from unwanted access. Two of the most reliable encryption algorithms out there are RSA and AES. AES is a symmetric encryption technique that uses a single key for both encryption and decryption, whereas RSA is an asymmetric encryption system that depends on two keys: a public key and a private key. These technologies are combined by "Zip Guard" to offer a complete data protection solution. Zip Guard not only protects data but also speeds up transfer times by compressing files before encryption. Its use of AES and Huffman coding allows for maximal compression with the least amount of risk to sensitive data. With the frequency of data breaches in the modern era, Zip Guard is a vital tool for safeguarding crucial information. This solution ensures efficiency and security for corporate data as well as personal files. Users may safely share and save data using Zip Guard without sacrificing function or privacy. By integrating these advanced techniques, Zip Guard establishes itself as a reliable and user-friendly tool for addressing modern data security and management challenges.

Motivation

This project is created in response to the growing demand for effective and safe data management solutions. Unauthorized access and data breaches are growing concerns along with the amount of digital information. For average users, traditional file compression and encryption techniques may be too complicated or challenging. By offering a straightforward interface that combines robust encryption settings with file compression, "Zip Guard" aims to make this process easier. The project satisfies the security requirements of a broad spectrum of users with its automatic key generation and customizable encryption options.

Additionally, the development of "Zip Guard" serves as an opportunity for our team to improve and test our programming skills. Working with advanced concepts like Huffman coding, RSA, and AES encryption in Core Java not only challenges our technical abilities but also helps us deepen our understanding of data security and software development practices. This project is a practical application of our learning, allowing us to enhance our coding proficiency while contributing to a real-world solution. The project builds collaboration and teamwork as we work together to overcome challenging issues with coding.

Our goal is to integrate user feedback into "Zip Guard" and make it able to satisfy increasing security requirements so that it will continue to be a reliable and useful tool in the future. We're sure that this experience will help us in the future when we work on projects that call for both software design and security skills. By designing an intuitive interface, we hope to ensure accessibility for both technical and non-technical users. Moreover, the use of Java-based encryption libraries allows us to delve deeper into secure coding practices. Enhancing the compression ratio with Huffman coding pushes our understanding of data structures and algorithmic efficiency. The versatility of this project aligns with industry needs, making it a standout portfolio addition. Lastly, "Zip Guard" underscores the importance of balancing functionality with usability in modern software solutions. Our efforts in creating "Zip Guard" also encourage a culture of innovation as we explore creative ways to improve existing data security methods. This project gives us an edge in understanding real-world application challenges, preparing us for future roles in software development and cybersecurity.

Related Work

Several projects and tools have addressed file compression and encryption, but few have combined these functionalities in a single package. Existing solutions like WinRAR and 7-Zip offer compression and basic encryption but lack the choice between advanced algorithms like RSA and AES. Academic research has explored the implementation of Huffman coding, RSA, and AES individually, but "Zip Guard" innovates by integrating these techniques into a cohesive tool. This project builds on previous work by offering a more versatile and user-friendly solution. "Zip Guard" not only offers more robust encryption solutions but also simplifies the user experience by automating several difficult tasks, including algorithm selection and key management. Customers may now enjoy speed, security, and ease of use all in one package, eliminating the need for them to pick between the three. Users can customize their security settings according to the sensitivity of their data thanks to the freedom to choose from a variety of encryption techniques. This combination makes "Zip Guard" a unique solution for secure and efficient data management.

Problem Identification

For file compression and encryption, numerous projects and tools have been created, all of which include advanced data management solutions. Although well-known programs like WinRAR and 7-Zip have simple encryption and compression capabilities, they are frequently rigid and difficult to use. These solutions often only offer one encryption type and don't have sophisticated features like RSA, which is necessary for asymmetric encryption. Furthermore, handling and storing keys by hand can be challenging when using these tools for key management.

"Zip Guard" overcomes these drawbacks in comparison to these current methods by including Huffman coding for compression and providing an option between RSA and AES encryption. Zip Guard's automatic key creation capability streamlines the procedure and increases accessibility for people without technical knowledge. "Zip Guard" is a more complete and user-friendly solution for safe data management than earlier initiatives because of its versatility, advanced security options, and simplicity of use.

Problem statement

In today's digital world, secure and effective data management is essential as the amount of sensitive data increases, so does the risk of unauthorized access and data breaches. Existing data management solutions, such as file compression tools and encryption utilities, often operate separately, complicating the process for the average user. Additionally, many of these tools lack the flexibility to choose from advanced encryption algorithms, limiting the user's control over their data security.

Users also face the challenge of selecting the right algorithm for their data and managing encryption keys. Moreover, the process of compressing files while maintaining their integrity and security is not always straightforward. As a result, many individuals and businesses are left with tools that compromise either usability or security. This creates a significant gap in the market for a tool that combines both file compression and encryption seamlessly, while also offering flexibility in encryption methods.

"Zip Guard" seeks to address these challenges by providing a user-friendly solution that integrates efficient file compression with robust encryption options. By offering a choice between Huffman coding for compression and AES or RSA encryption, the tool gives users complete control over their data security. This approach eliminates the complexity and limitations of existing solutions, ensuring that users can easily compress and encrypt their files with confidence.

Objective

- Develop a tool that combines efficient file compression using Huffman coding with secure encryption options.
- Provide users with a choice between RSA and AES encryption algorithms to meet varying security needs.
- Automate the key generation process to simplify encryption for users without technical expertise.
- Integrate Encryption, Decryption, Compression, and Decompression into a Single Package.

Methodology

The Zip Guard project begins with identifying the key requirements for secure file compression and encryption. We focus on creating a tool that allows users to compress files using the Huffman coding algorithm and encrypt them with either RSA or AES encryption algorithms. Java is chosen as the development language due to its extensive libraries and support for cryptographic functions, which are integral to the encryption process. The project begins with a thorough requirement analysis to identify the necessary functionalities, ensuring the solution addresses both compression and encryption needs. The goal is to create a tool that is efficient, secure, and easy to use for users at various technical levels.

For file compression, the system utilizes Huffman coding, a lossless compression algorithm that reduces the file size by analyzing the frequency of characters. The algorithm builds a frequency table and assigns variable-length codes to characters, optimizing storage space without sacrificing data integrity. This compression technique is integrated with the compression module of the application.

For encryption, the system supports two advanced methods: AES (Advanced Encryption Standard) and RSA (Rivest-Shamir-Adleman). AES is a symmetric key encryption algorithm that uses a single secret key for both encryption and decryption, while RSA is an asymmetric encryption system using a pair of keys (public and private). The user can select the desired encryption method based on their security requirements, offering flexibility and control over the level of encryption.

During key generation, the system automatically generates encryption keys for both AES and RSA. For AES, a random key is generated for each file, ensuring uniqueness and security. For RSA, a public-private key pair is created, providing robust encryption by using two keys: one for encryption and another for decryption. These keys are securely stored and managed by the system to ensure safe file handling.

The encryption and compression processes are seamlessly integrated, enabling users to compress and encrypt files in a single operation. The project ensures that files can be accurately decrypted and decompressed, without any loss of data, through the implementation of corresponding decryption and decompression algorithms.

Hardware Requirement

Hardware	Minimum Requirement
Processor	Intel Core i3 or higher
RAM	4 GB
Storage	500 MB or Above
Graphics	Integrated Graphics (eg., Intel HD)
Display	1366x768 resolution or higher
Input Device	Standard Keyboard and Mouse

Software Requirement

Software	Version/Details
Java Development Kit (JDK)	JDK 8 or higher
IDE	Visual Studio Code / IntelliJ IDEA (latest version)
Version Control System	Git (latest version)
Operating System	Windows 7 / macOS / Linux
Java Runtime Environment (JRE)	JRE 8 or higher

Results

Huffman Algorithm

```
test.txt
File Edit View
examplefile.com | Your Example Files.
examplefile.com | Your Example Files.
examplefile.com | Your Example Files.
examplefile.com | Your Example Files.
examplefile.com | Your Example Files.
examplefile.com | Your Example Files.
examplefile.com | Your Example Files.
examplefile.com | Your Example Files.
```

```
PS C:\Users\laksha\Desktop\MinorProjects> cd "C:\Users\laksha\Desktop\MinorProjects\" ; if ($?) { java ZipGuard.java } ; if ($?) { java ZipGuard.java }
```

ZIP GUARD

Welcome to Zip Guard - Your Secure File Management Tool!
With Zip Guard, you can compress and decompress files using Huffman coding, or encrypt and decrypt files with AES or RSA encryption.
Simply choose an option from the menu below, provide the required file paths, and let Zip Guard handle the rest.
Ensure your files are accessible, and keep your key files secure during encryption and decryption.

- 1: Compress a file using Huffman coding
- 2: Decompress a file using Huffman coding
- 3: Encrypt a file using AES
- 4: Decrypt a file using AES
- 5: Encrypt a file using RSA
- 6: Decrypt a file using RSA

Choice: 1
Enter the file path to compress:
./Sample/test.txt
File compressed successfully as: ./Sample/test_compressed.dat
PS C:\Users\laksha\Desktop\MinorProjects>

Sample

- test_compressed.dat
- test_tree.dat
- test.txt

test	13-12-2024 03:35	Text Document	1,025 KB
test_compressed	13-12-2024 07:30	DAT File	539 KB

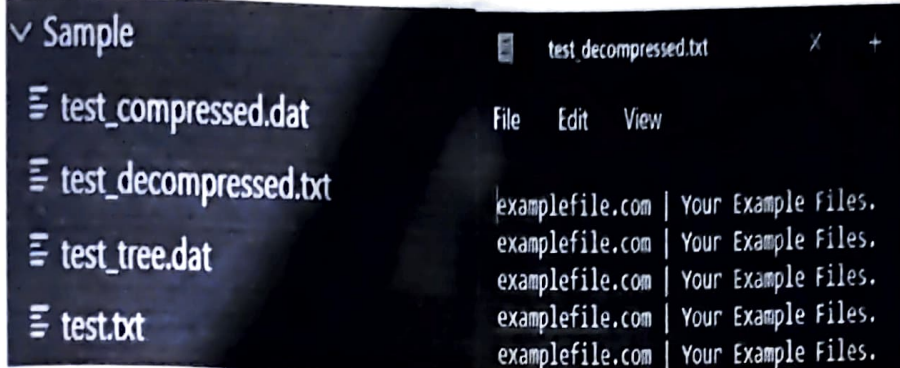
```
PS C:\Users\laksha\Desktop\MinorProjects> cd "C:\Users\laksha\Desktop\MinorProjects\" ; if ($?) { java ZipGuard.java } ; if ($?) { java ZipGuard.java }
```

ZIP GUARD

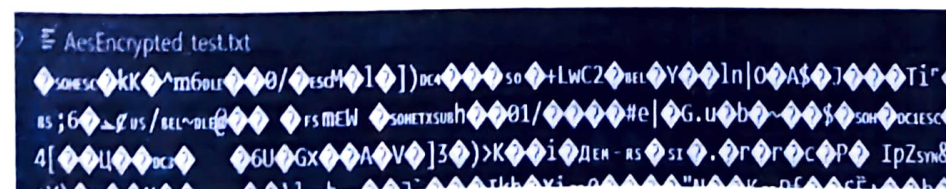
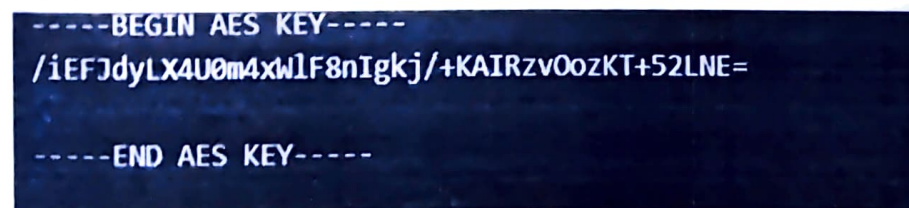
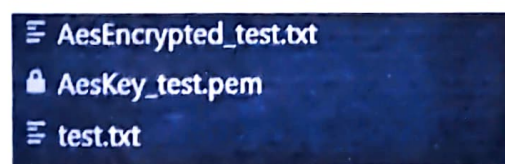
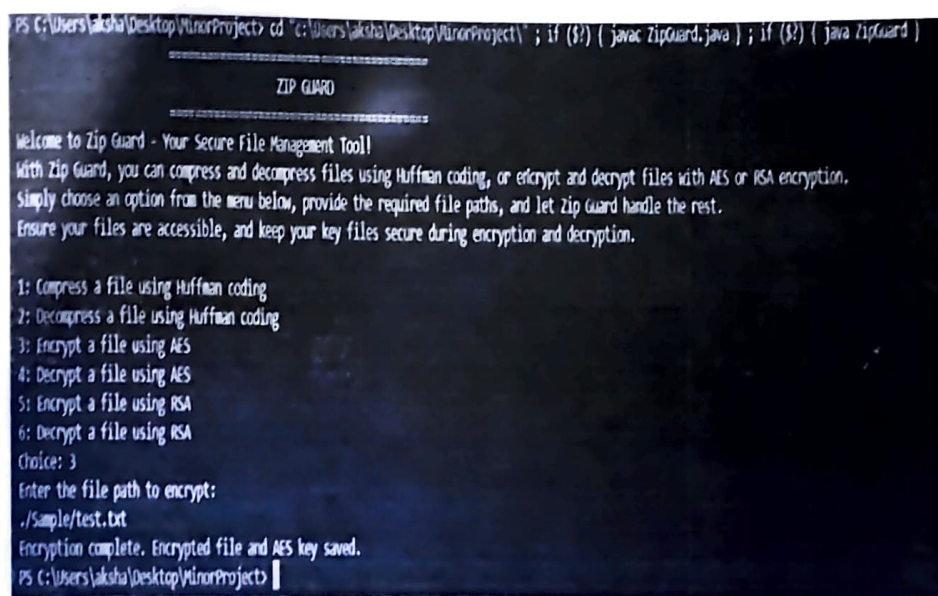
Welcome to Zip Guard - Your Secure File Management Tool!
With Zip Guard, you can compress and decompress files using Huffman coding, or encrypt and decrypt files with AES or RSA encryption.
Simply choose an option from the menu below, provide the required file paths, and let Zip Guard handle the rest.
Ensure your files are accessible, and keep your key files secure during encryption and decryption.

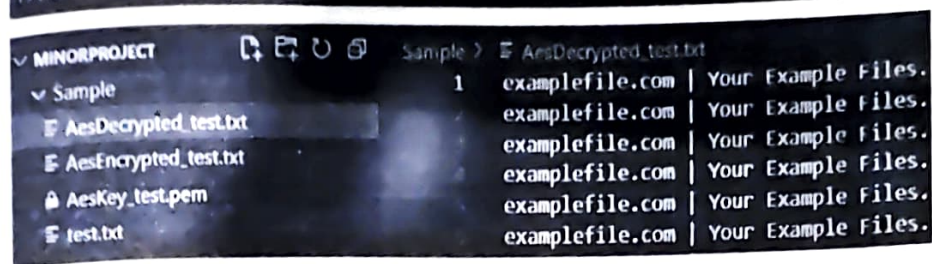
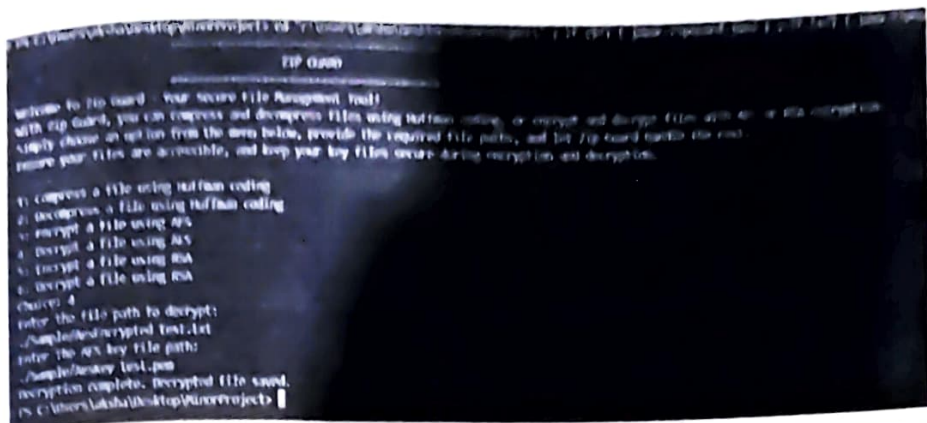
- 1: Compress a file using Huffman coding
- 2: Decompress a file using Huffman coding
- 3: Encrypt a file using AES
- 4: Decrypt a file using AES
- 5: Encrypt a file using RSA
- 6: Decrypt a file using RSA

Choice: 2
Enter the compressed file path: ./Sample/test_compressed.dat
Enter the tree file path: ./Sample/test_tree.dat
File decompressed successfully as: ./Sample/test_decompressed.txt
PS C:\Users\laksha\Desktop\MinorProjects>

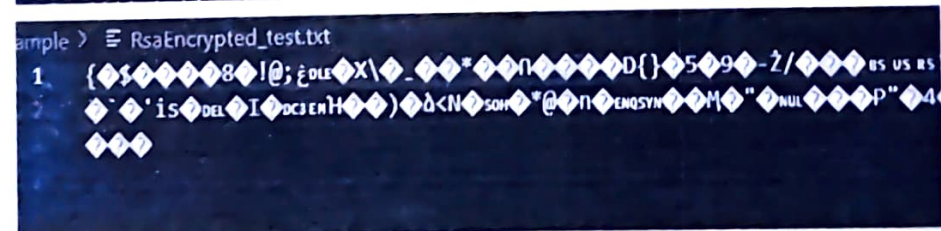
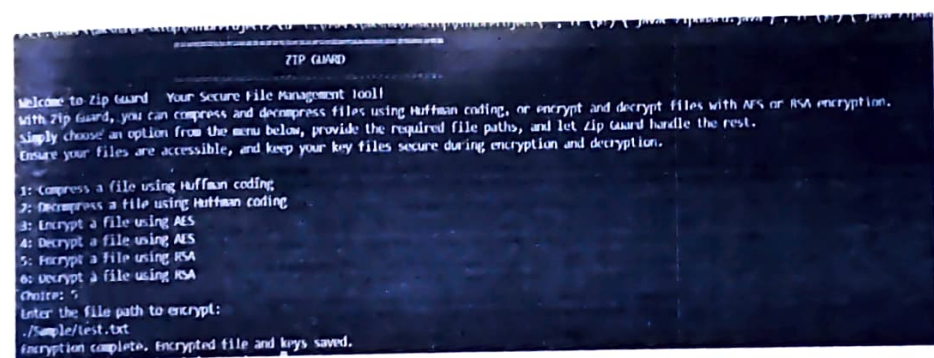
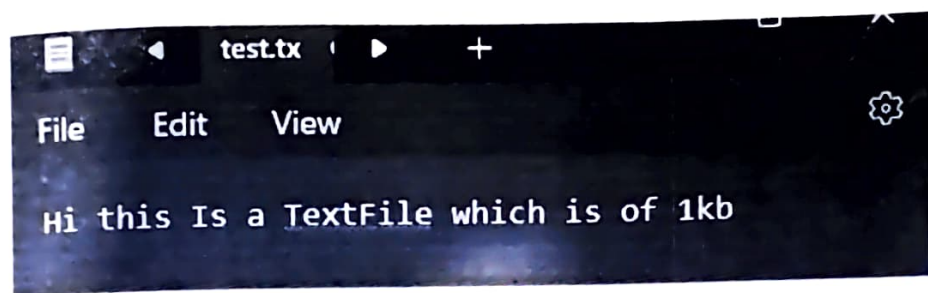


AES Algorithm:





RSA Algorithm



[illegible]

END PRIVATE KEY

MI1BIJANBgkqhkiG9w0BAQEFAAOCAQIAMIIBcRKCAQEA/DZmN1RQlpbjjmL2GJ0
 47Bq0/wskZn9o0p19ggJbahrHtLtsVb3YXqN4o4Tae1wA++yxpjh2MKY3Juxf
 17yepCm1yTET+QxqY73Jsgk7hV1XtPtphtcmQuf7gqxJesbuIV1g8ew/Y/z
 3dgrCufw7wDw9Y5Gh61eDPUZnuzt1br4/rP5sa0uyv9Vmqdgyf12G7fC6e
 3o9kaofp39+530tC68fwYgLaZyG1Zrws1oV6XedqYz3Dbkapudpft1IqiveMm
 7V1rqlah13Zi17P0JrhZ1mQdYQJEEY0S5Tkaw1/Dy4JazNGLHioj1LZDK3hzo
 BOTDAN

-END PUBLIC KEY-----

- RsaEncrypted_test.txt
- RsaPrivateKey_test.pem
- RsaPublicKey_test.pem
- test.txt

```
ms C:\Users\kshu\Desktop\WinrarProject> cd "C:\Users\kshu\Desktop\WinrarProject"; if ($?) { javac ZipGuard.java }; if ($?) { java ZipGuard
```


ZIP GUARD

License: 10-ZipGuard - Your Secure File Management Tool!

With Zip Guard, you can **compress** and **decrypt** files using Huffman coding, or encrypt and decrypt files with AES or RSA encryption.

Simply choose an option from the menu below, provide the required file paths, and let Zip Guard handle the rest. Ensure your files are accessible, and keep your key files secure during encryption and decryption.

```
1: Compress a file using Huffman coding
2: Decompress a file using Huffman coding
3: Encrypt a file using AES
4: Decrypt a file using AES
5: Encrypt a file using RSA
6: Decrypt a file using RSA
Choice: 6
Enter the file path to decrypt:
c:\msdcs\unscripted\cat.txt
Enter the private key file path (in PEM format):
c:\msdcs\unscripted\cat.pem
Decryptive function: Decrypt file saved
c:\msdcs\unscripted\cat.txt
```



File Explorer window showing the contents of the 'Sample' folder. The files listed are:

- RsaDecrypted_test.txt
- RsaEncrypted_test.txt
- RsaPrivateKey_test.pem
- RsaPublicKey_test.pem

The selected file, 'RsaDecrypted_test.txt', contains the text:

```
Hi this is a Textfile under 245 bytes
```


References

[1] Oracle (2024) "Java™ Platform, Standard Edition Security Developer's Guide: Standard Names," Available at:

<https://docs.oracle.com/en/java/javase/22/docs/specs/security/standard-names.html>

[2] Oracle (2024) "Java™ SE 8: Cryptography Roadmap," Available at:

<https://www.java.com/en/jre-jdk-cryptoroadmap.html>

[3] Data Structures Project - File Compression Using Huffman Coding in JAVA

(2023) "YouTube," Available at: <https://www.youtube.com/watch?v=S0Wua5WxKZI>

[4] IETF (2016) "PKCS #1: RSA Cryptography Specifications Version 2.2," RFC 8017. Available at: <https://www.ietf.org/rfc/rfc8017.html>

[5] Schneier, B. (1996) Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2nd ed. New York: John Wiley & Sons. Available at:

<https://mrjacse.wordpress.com/wp-content/uploads/2012/01/applied-cryptography-2nd-ed-b-schneier.pdf>

Approved By

(Mentor)

(Panel Member)