

VISVESVARAYATECHNOLOGICALUNIVERSITY BELAGAVI



Project Report on

“BIOMETRIC DIGITAL SIGNING AND DOCUMENT SHARING USING IPFS (INTERPLANETARY FILE SYSTEM)”

Submitted in the partial fulfillment for the requirements of the degree of

BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING

Submitted By

PRAJODH PRAGATH SUNDER	1BY19CS104
PRATHAM H SUNNAL	1BY19CS109
RAKSHITH GOWDA A	1BY19CS114
ROHAN JOY A	1BY19CS122

Under the guidance of
DR. RADHIKA K R

ASSISTANT PROFESSOR,
Department of CSE,
BMSIT&M



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064.
2022-2023

VISVESVARAYATECHNOLOGICALUNIVERSITY BELAGAVI

BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
YELAHANKA, BENGALURU – 560064

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the Project work entitled “**BIOMETRIC DIGITAL SIGNING AND DOCUMENT SHARING USING IPFS**” is a bonafide work carried out by **Prajodh pragath sunder (1BY19CS104), Pratham H Sunnal (1BY19CS109), Rohan Joy A (1BY19CS122), Rakshith Gowda A (1BY19CS114)**, in partial fulfillment for the award of **Bachelor of Engineering Degree in Computer Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2022-2023. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report. The project report has been approved as it satisfies the academic requirements in respect of project work for B.E. Degree.

Signature of the Guide

Dr. Radhika K R
Assistant Professor
Dept. of CSE, BMSIT&M

Signature of the HOD

Dr. Thippeswamy G
Professor & HOD,
Dept. of CSE, BMSIT&M

Signature of Principal

Dr. Mohan Babu G N
Principal, BMSIT&M

External VIVA-VOCE

Name of the Examiners

- 1.
- 2.

Signature with Date

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and every one who has helped us make this project a success.

We heartily thank our Principal, **Dr. MOHAN BABU G N, BMS Institute of Technology and Management**, for his constant encouragement and inspiration in taking up this project.

We heartily thank our Head of the Department, **Dr. Thippeswamy G, Department of Computer Science and Engineering, BMS Institute of Technology and Management**, for his constant encouragement and inspiration in taking up this project.

We gracefully thank our Project Guide, **Dr. Radhika K R, Assistant Professor, Department of Computer Science and Engineering**, for her guidance, support, and advice.

Special thanks to all the teaching and non-teaching staff members of Computer Science and Engineering Department for their help and kind co-operation.

Lastly we thank our parents and friends for the support and encouragement given to us in completing this precious work successfully.

Prajodh Pragath Sunder 1BY19CS104

Pratham H Sunnal 1BY19CS109

Rakshith Gowda A 1BY19CS114

Rohan Joy A 1BY19CS122

ABSTRACT

There is a long and hectic process involved in getting the documents verified by the higher-ups and the documents undergo a series of changes and modifications before getting the final approval. Our proposed system provides a secure way to exchange, verify, modify, and comment out the changes performed on the document. The process begins with the creation of a digital document by a user. The document is then encrypted and stored on the IPFS. The user can then share the document with other authorized parties by using their unique ID's. When a party receives the document, they can verify its authenticity and contents using the cryptographic hash of the document stored on the IPFS. If the document meets their requirements, they can then sign and encrypt it using their private key. The signed document is then stored on the IPFS and made available to other authorized parties.

TABLE OF CONTENTS

ACKNOWLEDGEMENT		i
ABSTRACT		ii
TABLE OF CONTENTS		iii-iv
LIST OF FIGURES		v
LIST OF TABLES		vi
Sl No	Chapter Name	Page No
1	Introduction	1
1.1	Background	2-4
1.2	Literature Survey	5-6
1.2.1	Limitations of Existing System	7
1.2.2	Why IPFS?	8-10
1.3	Motivation	10-11
1.4	Problem Statement	11
1.5	Aims and Objectives	11
1.6	Scope	12
1.7	Challenges	12-13
1.8	Organization of the thesis	13-14
2	Overview	15-17
2.1	Input	16
2.2	Tools	17
2.3	Procedure	17
2.4	Output	17
3	Requirement Specification	18-20
3.1	Functional Requirements	18-19
3.2	Non-Functional Requirements	19
3.3	Software Requirements	20
3.4	Hardware Requirements	20

4	Detailed Design	21-24
4.1	Overview	21
4.2	Use Case Diagram	22
4.3	Sequence Diagram	23-24
5	Implementation	25-31
5.1	Programming Language	25
5.2	Frameworks	26
5.3	Algorithms	26-29
5.3.1	Why Elliptical Curve Algorithm?	30-31
6	Testing	32-35
6.1	Registration Page Testing	32
6.2	Face Recognition Testing	32-33
6.3	File Upload Testing	33
6.4	Digital Signature Testing	33-34
6.5	File Encryption and Decryption Testing	34-35
7	Experiment Results	36-61
7.1	Code Pack	36-49
7.2	Results with Screenshots	50-61
8	Conclusion	62
	References	63-65

LIST OF FIGURES

FIGURE NO	CAPTION	PAGE NO
Figure 4.1	Overview of the System	21
Figure 4.2	Use Case Diagram	22
Figure 4.3	Sequence Diagram	23
Figure 5.1	Key Size Comparison	30
Figure 7.1	Code snippet of file upload function	36
Figure 7.2	Code snippet of generate keys function	38
Figure 7.3	Code snippet of file encryption using RSA Algorithm	39
Figure 7.4	Code snippet of AES encryption	40
Figure 7.5	Code snippet of file approval function	42
Figure 7.6	Code snippet of file decrypt function	43
Figure 7.7	Code snippet of the digital signature function	45
Figure 7.8	Code snippet of file verification	46
Figure 7.9	Code snippet of face recognition module	47
Figure 7.10	Code snippet to capture the streaming video	48
Figure 7.11	Website welcome page	50
Figure 7.12	Registration Page	51
Figure 7.13	Image Registration	52
Figure 7.14	Generate your private key	53
Figure 7.15	Login page	54
Figure 7.16	Face Recognition	55
Figure 7.17	Profile page	56
Figure 7.18	File upload page	57
Figure 7.19	Received files	58
Figure 7.20	Adding comments to the file	59
Figure 7.21	Approving the file using digital signature	59
Figure 7.22	Checking the document status	60
Figure 7.23	Valid Document	61
Figure 7.24	Invalid Document	61

LIST OF TABLES

TABLE NO	CAPTION	PAGE NO
Table 1.1	Literature Survey	5-6

CHAPTER 1

INTRODUCTION

The digital exchange of documents has become prevalent in today's world, necessitating secure and efficient file sharing systems. However, conventional methods often suffer from security vulnerabilities, slow speeds, and reliance on personal information. The Interplanetary File System (IPFS) offers a decentralized and secure solution for digital document exchange.

IPFS operates on a peer-to-peer network architecture, where files are encrypted, split into chunks, and distributed across multiple nodes. This decentralized approach ensures the security and integrity of files, even if one node is compromised. By utilizing IPFS for digital document exchange, users can securely store and share documents without relying on centralized servers or third-party services. This enhances security, privacy, and user control over data.

The concept of digital exchange and biometric signing using IPFS combines the security and decentralization of IPFS with the accuracy and authenticity of biometric signatures. Biometric signatures leverage unique physical characteristics like fingerprints, facial recognition, or voice recognition to create digital signatures tied to individuals. This provides a highly secure method of identity verification, as biometric features are difficult to replicate or forge.

Facial recognition-based digital signing is a modern approach that combines the convenience of digital signatures with the accuracy and security of facial recognition technology. By utilizing facial features, a digital signature can be generated and associated with the individual, ensuring secure and reliable document signing while preserving authenticity and privacy.

The integration of biometric signatures with IPFS enhances the security and verification of digital document exchange. By employing biometric signatures for document signing on IPFS, the identity of the signatory can be accurately and verifiably confirmed, mitigating fraud and fostering trust in digital transactions.

1.1 Background

IPFS (Interplanetary File System) is a decentralized file storage system that provides a new way of storing and sharing files across a distributed network of computers. Unlike traditional file storage systems, IPFS doesn't rely on a central server to store files. Instead, files are stored across a network of computers, called nodes. The IPFS network is designed to be highly resilient and fault-tolerant, allowing documents to be shared even if some nodes in the network go offline. This makes it an ideal solution for organizations and businesses that require a high level of data redundancy and availability.

IPFS uses content-addressing, which means that each document is uniquely identified by its content, rather than its location on the network. This allows documents to be stored and shared more efficiently, since identical documents are only stored once, even if they are shared by multiple users. When a document is shared using IPFS, it is encrypted and broken down into smaller chunks, which are then distributed across the network. Each chunk is assigned a unique identifier, which is used to retrieve the chunk from the network when the document is accessed.

IPFS allows users to host and receive content. It is built around a decentralized system of user-operators who hold a portion of the overall data, creating a resilient system of file storage and sharing. Any user in the network can serve a file by its content address, and other peers in the network can find and request that content from any node who has it using a distributed hash table (DHT). IPFS also supports versioning, allowing multiple versions of a document to be stored and accessed. This can be useful for tracking changes to a document over time, or for keeping a record of previous versions of a document.

Digital exchange of documents using IPFS is a modern approach to exchanging and sharing digital documents securely and efficiently. To use IPFS for digital document exchange, users can upload their documents to the IPFS network, which stores the documents across multiple nodes, making it highly resistant to data breaches and hacking. This approach ensures that the documents are more secure and reliable than using traditional centralized servers. To access the documents stored on IPFS, users can use a unique identifier known as a content address, which is generated based on the content of the file.

A biometric digital signature is a method of digitally signing a document using biometric authentication, such as facial recognition. This approach ensures that the signature is unique to the signer and cannot be replicated or forged by anyone else. To use biometric digital

signatures for document signing, the signer must first create a digital signature template, which contains their biometric information. This can include facial features, voice recognition, or other unique biometric identifiers. When the signer wishes to sign a document, they are prompted to verify their identity using their biometric information.

Biometric signing of documents using facial recognition is a secure and efficient way to authenticate the identity of a signer and ensure that a document is legally binding. Facial recognition uses unique facial features to verify the identity of a signer, which can provide a higher level of security compared to traditional electronic signatures. The use of facial recognition for biometric signing is becoming increasingly popular in various industries, including banking, healthcare, and legal services.

Facial recognition can be used to sign a variety of documents, such as contracts, legal agreements, and consent forms. They can also be used for remote signing, allowing signers to sign documents from anywhere in the world. The process of biometric signing using facial recognition typically involves several steps. First, the signer's face is captured and stored securely. This can be done using a camera on a mobile device or computer, or through a dedicated facial recognition device. Next, the facial recognition system analyses the unique facial features of the signer and compares them to a pre-existing database of faces. This comparison is done using algorithms that are designed to identify and match specific facial features, such as the distance between the eyes, nose, and mouth.

In this work, we have made use of the facial recognition as the means to generate the biometric data and generate the private keys using the same. There were two considerations as to why we made use of biometric data (face). First, it enhances the security of the generated keys as a person's biometric data is unique to that person and cannot be changed. Second, to randomize to generation of the generated private key as people are still able to hack into the cryptographic module where the keys are stored and generate the private key. Now using this private key, the receiver or the concerned authority can apply his digital signature on the document.

The digital signature is then stored securely along with the document, ensuring that the document is legally binding and that the identity of the signer has been verified. To verify the digital signature, the facial recognition system analyses the signature and compares it to the original signature captured during the signing process. This comparison is done using algorithms that are designed to identify and match specific facial features, such as the distance between the eyes, nose, and mouth.

Facial recognition can also be used to verify the identity of the person accessing a document. For example, if a document contains sensitive information, it may be necessary to ensure that only authorized individuals are able to access it. By using facial recognition to verify the identity of the person accessing the document, businesses and organizations can increase the security and confidentiality of their sensitive information. If any changes are needed the receiver can send comments back to the sender to make the changes and send the document again.

Combining IPFS with biometric signing can provide a powerful tool for secure digital exchange of documents. When a document is shared using IPFS, it can be signed using biometric authentication to ensure that the document is only accessible by authorized parties. This can reduce the risk of data breaches and unauthorized access to sensitive information.

1.2 Literature Survey

Table 1.1

No	Paper Title	Publications	Technology/Tools used	Contributors
1	A Dynamic Resource Management Scheme for Content-Based P2P Networks: A Case Study of IPFS [1].(2021)	Published in IEEE Access. Bo Fu, et al. Switzerland	Load-Balancing Caching Replication Adaptive resource allocation	The authors evaluate the performance of the scheme through simulations and experiments on a real IPFS network and demonstrate its effectiveness in improving system scalability and reducing resource waste.
2	Exchanging Digital Documents Using Blockchain Technology [2]. (2021)	Proceedings of the 3 rd International Conference on Electrical, Communication and Computer Engineering (ICECCE). Anas Abu Talab et al. Jordan	Developed a decentralized system for exchanging digital documents, using Ethereum blockchain, Interplanetary File System (IPFS) Smart contracts and AES algorithm to overcome the problems of using trusted third parties in exchanging documents.	The model has satisfied the three main security requirements which are confidentiality, integrity, and availability. The model was affordable (costed 0.0552\$ per transaction)

3	Blockchain based Framework for Student Identity and Educational Certificate Verification [4]. (2021)	Proceedings of the 2 nd Int. Conference on Electronics and Sustainable Communication Systems (ICESC-2021) Aastha Chowdhary et al Suratkal	Developed using Ganache-cli (a local Blockchain), Truffle, Metamask and IPFS.	They were able to link the certificates to the user Identity for more security and to develop a foul-proof system.
4	Software Development of Electronic Digital Signature Generation at Institution Electronic Document Circulation [7]. (2020)	IEEE East-West Design & Test Symposium (EWDTS) Olga A. Safaryan et al. Russia	Implemented in Python. Development environment used is Microsoft Visual Studio Code. The graphical interface of the software tool is implemented as web page using HTML, CSS and JS. Made use of Blockchain and PKI.	The authors were the first to use blockchain technology for remote cloud generation and verification of electronic signatures. The elements were designed by keeping scalability and extensibility in mind.
5	Management of digital documents with encrypted signature, through the use of centralized PKI, and distributed using blockchain for a secure exchange[6]. (2019)	Journal of Research and Development. 2019, 5-15: 26-37. Juan Carlos Olivares Rojas et al. Mexico	Made use of RSA algorithm to encrypt the document. Java SE to implement the digital signature. MySQL to manage data. Blockchain to provide decentralization and added security.	It was observed that the proposed software can be used reliably to protect the integrity of the data. Verification methods for the generation of digital signature were found to be effective. And generation of the same was relatively fast.

1.2.1 Limitations of Existing System

While the existing systems do provide some level of functionality and services, it doesn't include all the features we are trying to embed in our work. Some of the limitations are:

1. **Centralized Architecture:** Many existing systems for digital exchange and signature of documents rely on a centralized architecture, where a central authority is responsible for verifying and signing documents. This architecture is prone to single points of failure and is vulnerable to attacks and tampering .
2. **Security:** They might lack security which can be detrimental when handling sensitive subjects or documents.
3. **Lack of Biometric Authentication:** Many existing systems rely on traditional username/password authentication or cryptographic keys for user verification. This method is susceptible to password breaches or key compromises.
4. **Lack of Tamper-proof Document Signing:** Existing systems may provide a digital signature mechanism, but they may not offer robust protection against tampering or alteration of signed documents.
5. **Limited Scalability and Performance:** Some existing systems may experience performance bottlenecks or scalability issues when dealing with a large volume of documents or user requests.
6. **Modification:** Few of the existing paper only authenticates the documents but doesn't provide any facility to modify the document.
7. **Use of Blockchain:** Many proposed systems use blockchain for file transfer which can be computationally and economically costly.
8. **Limited Interoperability:** Some existing systems are not interoperable with other systems, making it difficult for users to exchange documents across different platforms.
9. **Dependence on Third-Party Services:** Some existing systems rely on third-party services, such as cloud storage providers and digital signature providers. This dependence can result in increased costs and potential security risks.
10. **Most of the existing systems aren't providing a service which integrates all the goals we are trying to achieve; editing file, sharing it securely using IPFS and provide the feature to send comments/notes to make sure that everyone is working on the same document and structure the perfect document.**

1.2.2 Why IPFS?

IPFS is a decentralized file system that enables the sharing and storage of files in a distributed manner, using content addressing to identify and retrieve files based on their content. It offers numerous benefits, including decentralized storage, content addressing, data integrity and authenticity, efficient content distribution, peer-to-peer communication, offline file access, versioning and deduplication, community and ecosystem support, cost-effectiveness, and independence from third-party services. These advantages make IPFS an attractive option for secure and efficient document exchange and signature, as it eliminates the risks associated with centralized systems and offers enhanced privacy, scalability, security, and interoperability. IPFS provides a promising solution for various applications such as web hosting, data sharing, and distributed applications. In contrast, blockchain is a distributed ledger technology that enables secure and transparent transactions between parties without the need for a trusted third party, commonly used for applications such as cryptocurrency, supply chain management, and digital identity.

Reasons to consider:

1. **Decentralized Storage:** IPFS provides a decentralized storage solution, where files are distributed across multiple nodes in a peer-to-peer network. This decentralized approach eliminates the reliance on a central server, reducing the risk of single points of failure and enhancing data availability and resilience. In the context of document exchange, this ensures that documents are not stored in a single vulnerable location, enhancing security and reliability.
2. **Content Addressing:** IPFS uses content addressing, meaning files are identified by their content rather than their location. Each file is assigned a unique cryptographic hash based on its content, enabling efficient retrieval and verification. This content-based addressing also ensures that files remain immutable, as any changes to the content will result in a different hash.
3. **Data Integrity and Authenticity:** Since IPFS is making use of content addressing (where files are identified by their content hash), this ensures that files remain immutable and tamper-proof. Any modifications to a document will result in a different hash, immediately indicating a mismatch. This feature ensures the integrity and authenticity of documents, which is crucial for digital signing and maintaining the trustworthiness of exchanged files.
4. **Efficient Content Distribution:** IPFS allows for efficient content distribution through its peer-to-peer architecture. When a document is shared, it is automatically

replicated across the network, enabling faster and more efficient retrieval for subsequent users. This can be particularly beneficial in scenarios where multiple parties need access to the same document, ensuring faster document exchange and reducing the burden on centralized servers.

5. **Efficient Content Distribution:** IPFS allows for efficient content distribution through its peer-to-peer architecture. When a document is shared, it is automatically replicated across the network, enabling faster and more efficient retrieval for subsequent users. This can be particularly beneficial in scenarios where multiple parties need access to the same document, ensuring faster document exchange and reducing the burden on centralized servers.
6. **Peer-to-Peer Communication:** IPFS utilizes peer-to-peer communication protocols, enabling direct communication between nodes without the need for intermediaries. This peer-to-peer architecture promotes efficient and fast file sharing, eliminating bottlenecks that may arise in client-server models.
7. **Offline File Access:** IPFS supports offline file access, as nodes can store and retrieve files even when not connected to the internet. Files retrieved and cached from IPFS remain accessible offline as long as the original content remains available within the local network.
8. **Versioning and Deduplication:** IPFS employs content-addressed storage, enabling efficient versioning and deduplication of files. When multiple versions of a file are stored, only the differences are added to the network, reducing storage requirements. Similarly, identical files shared by different users are deduplicated, saving space and network bandwidth.
9. **Community and Ecosystem:** IPFS has gained significant community support and adoption, leading to the development of a thriving ecosystem of tools, libraries, and applications built on top of the IPFS protocol. This active community contributes to the continuous improvement, expansion, and innovation within the IPFS ecosystem.
10. **Cost:** IPFS is generally less expensive to use for file sharing compared to blockchain. In blockchain, transactions usually require a fee to be paid to miners to confirm the transaction.
11. **Independence from Third-Party Services:** IPFS is designed to be independent from third-party services, such as cloud storage providers and digital signature providers. This independence reduces costs and potential security risks, while also giving users greater control over their files.

Overall, IPFS offers a number of advantages over existing systems for digital exchange and signature of documents. Its decentralized architecture, enhanced privacy, scalability, security, interoperability, and independence from third-party services make it a promising solution for secure and efficient document exchanges.

1.3 Motivation

The motivation to pursue this project stems from the need for a more secure, reliable, and decentralized way to share and store files. Traditional file sharing and storage systems rely on centralized servers, which can be vulnerable to security breaches, downtime, and other issues. Some of the factors are:

1. **Security:** In today's digital world, security is of utmost importance. With the increasing instances of cyber-attacks and data breaches, there is a need for more secure and reliable systems for digital document exchange and signing. Biometric authentication provides an added layer of security by verifying the identity of the signer using unique biological traits, such as facial recognition, which are difficult to replicate or forge.
2. **Efficiency:** Traditional document signing processes can be time-consuming and require physical presence. With a biometric digital signing and exchange system, the entire process can be done remotely, saving time and resources.
3. **Cost Reduction:** By eliminating the need for physical presence and reducing the amount of paperwork involved in document signing, a biometric digital signing and exchange system can help reduce costs associated with document signing processes.
4. **Accessibility:** A biometric digital signing and exchange system can make document signing and exchange more accessible to individuals who are physically unable to sign documents or are located in remote areas where physical signing may not be feasible.
5. **IPFS Technology:** IPFS technology offers a decentralized and secure way to store and share documents, making it an ideal platform for digital document exchange and signing. By leveraging IPFS, the system can be more secure, efficient, and cost-effective.

Overall, the motivation behind building a project for biometric digital signing and exchange of documents using IPFS is to provide a more secure, efficient, and accessible way for individuals and organizations to sign and exchange documents in the digital age.

1.4 Problem Statement

The problem addressed in this project is the need for a secure and efficient system for document exchange and digital signing. Existing systems often lack sufficient security measures, reliable authentication mechanisms, efficient transmission methods and have a single point of failure. This leads to vulnerabilities such as unauthorized access, data breaches, compromised document integrity, weak authentication and inefficient document exchange processes. Additionally, the absence of biometric authentication in the document signing process increases the risk of identity fraud. The project aims to develop a solution that incorporates biometric digital signing and utilizes IPFS for secure and decentralized document storage. The solution should ensure the authenticity, integrity, and confidentiality of exchanged documents while providing a user-friendly interface for seamless and reliable document exchange.

1.5 Aims and Objectives

The Objectives are as follows:

1. To survey research papers and learn about existing projects related to our project topic.
2. To discover the research gaps and shortcomings of the existing systems.
3. To propose a system which addresses these shortcomings and adds additional functionality.
4. To develop a project which adheres to the proposed system.
5. To evaluate the performance and effectiveness of the developed system through rigorous testing and analysis, comparing it to existing systems and benchmarking against relevant metrics and criteria.

1.6 Scope

The scope of this project for biometric digital signing and exchange of documents using IPFS includes the following:

1. Development of a user-friendly web application for document signing and exchange using facial recognition biometric authentication.
2. Integration of the IPFS decentralized storage system for secure and reliable storage and sharing of signed documents.
3. Development of a secure and efficient biometric authentication system for facial recognition using modern computer vision techniques.
4. Implementation of digital signature technology for secure and tamper-proof signing of documents.
5. Implementation of end-to-end encryption for secure communication between users and the application.
6. Testing and evaluation of the system to ensure its security, reliability, and user-friendliness.

The project will focus on using facial recognition biometric authentication for identity verification during the document signing process. The system will be built on the IPFS platform to ensure the security and privacy of the signed documents.

The project will also include the development of a user-friendly web application that can be accessed from any device with an internet connection. The application will enable users to upload documents for signing and securely share the signed documents with other users.

Overall, the project aims to provide a secure, efficient, and accessible solution for document signing and exchange using biometric authentication and IPFS technology.

1.7 Challenges

Building any project consumes lot of time and presents many challenges. Some of the challenges we faced are:

1. The design of the project architecture presented a significant problem for us. To design our own, we had to employ current technologies and conduct a thorough review of the literature.
2. The following step was to choose wisely the programming language and the technology we would apply.

3. Task distribution and allocation among team members to ensure an efficient and seamless development cycle presented another problem.
4. Learning about server-side events was a hurdle when streaming video to a Django url.
5. We needed to understand how to manage the dlib requirement in order to use the face_recognition module for the model.
6. Choosing an appropriate encryption standard was difficult.
7. Because we experienced difficulty with the length of the private key, we employed public key encryption on a file hash rather than the entire file. By doing this, we were able to save the time required to encrypt the entire file.
8. Because RSA has more substantial private and public keys, we had to survey and research the alternative options, which was time-consuming. In the end, we decided to create the keys using the Elliptic curve algorithm because it offers security that is quite close to that of the RSA algorithm.

Overcoming these challenges requires a deep understanding of the technologies involved, a rigorous design and development process, and thorough testing and validation. It also requires collaboration between experts in different areas, such as computer vision, distributed systems, and cybersecurity, to ensure that all aspects of the system are properly addressed.

1.8 Organization of the thesis

The organization of the thesis for this project on biometric digital signing and exchange of documents using IPFS will follow a standard structure, which includes the following sections:

1. Chapter 1 – Introduction: An outline of the project's goals, drivers, and scope is mentioned in this chapter. The literature survey and the shortcomings in the current system are also included. The difficulties encountered during project construction are also mentioned.
2. Chapter 2 – Overview: This chapter discusses the project's overall process, its input and output, and the steps taken. It also includes the tools used to build the project.
3. Chapter 3 – Requirements Specification: The project's functional and non-functional needs are discussed in this chapter. Additionally, it covers the software and hardware needed to develop the project.

4. Chapter 4 – Detailed Design: The system architecture is a prominent topic of discussion in this chapter. It also discusses the various UML (Unified Modelling Language) diagrams, including use case diagrams and sequence diagrams.
5. Chapter 5 – Implementation: In this chapter, we talk about the programming language and the frame work used to build the project. Additionally, we go over the algorithms we utilise and why.
6. Chapter 6 – Testing: The main areas of discussion in this chapter are the many sorts of testing carried out, such as the File upload tests, Face recognition tests, Registration page tests, digital signature tests, and lastly the File encryption and decryption tests.
7. Chapter 7 – Experimental Results: Every page of the website is illustrated in a pictorial depiction. We also go over the outcomes we've gotten, along with screen shots of each input and the resulting output.
8. Chapter 8 – Conclusion: This chapter will summarize the key findings of the project and provide a conclusion on the project's success in meeting its objectives.
9. References: This section will provide a list of all the references cited in the thesis.

The thesis will be structured in a logical and organized manner to ensure that the reader can follow the development process, understand the rationale behind the choices made, and evaluate the effectiveness of the final system.

CHAPTER 2

OVERVIEW

The project aims to develop a comprehensive system for the secure and efficient biometric digital signing and exchange of documents using IPFS. It addresses the limitations of existing systems by incorporating robust biometric authentication, decentralized storage, and encryption mechanisms to ensure the authenticity, integrity, and confidentiality of the exchanged documents.

The project's objectives include implementing a robust biometric digital signing mechanism, leveraging facial recognition technology for authentication and ensuring the authenticity and integrity of document signings. Additionally, the system will utilize IPFS for decentralized storage, taking advantage of its content addressing and distributed file storage capabilities to provide a secure and reliable platform for document exchange.

Key Components and Features:

1. **Biometric Digital Signing:** The system will integrate facial recognition technology to provide biometric authentication for document signing. Users will be able to securely sign documents using their unique facial biometrics, ensuring the authenticity of the signatures.
2. **IPFS Integration:** The system will leverage the decentralized and distributed nature of IPFS for document storage and retrieval. Each document will be assigned a unique content hash, allowing for efficient and reliable sharing and retrieval across the network. IPFS's content addressing ensures the integrity and immutability of the documents.
3. **Secure Document Exchange:** The system will provide a secure platform for document exchange, ensuring that only authorized users can access and exchange documents. Encryption algorithms, such as Fernet (AES), will be used to encrypt the documents during transmission, protecting them from unauthorized access and maintaining their confidentiality.
4. **User-Friendly Interface:** A web-based interface will be developed to facilitate easy document uploading, signing, and retrieval. The interface will be designed to be intuitive, user-friendly, and accessible to users with varying levels of technical expertise.

5. **Testing and Evaluation:** Rigorous testing and evaluation will be conducted to validate the functionality, security, and performance of the system. Various test scenarios, including document exchange, biometric signing, and system resilience, will be performed. User feedback and performance metrics will be collected to assess the system's effectiveness and user satisfaction.
6. **Scalability and Future Extensibility:** The system will be designed with scalability in mind, allowing it to handle increasing user demands and document volumes. The architecture will be modular, enabling future enhancements and integration with other systems or technologies.

Throughout the project, comprehensive testing and evaluation will be conducted to validate the system's functionality, security, and performance. This will include testing various scenarios and evaluating the system's effectiveness, efficiency, and user satisfaction. Scalability and extensibility will also be considered, ensuring the system can handle increased user demands and allow for future enhancements and integration with other technologies.

By the end of the project, the aim is to deliver a robust, secure, and user-friendly system that facilitates the biometric digital signing and exchange of documents. The system's utilization of IPFS ensures decentralized and secure document storage, while biometric authentication enhances the overall security and trustworthiness of document signings. The project strives to address the limitations of existing systems and provide a reliable and efficient solution for secure document exchange in various domains.

The project's general process, including its inputs, tools employed in its development, overall procedure, and anticipated output, is covered in the parts that follow.

2.1 Input:

The following is supplied as input to the project.

1. Username and Password (At the time of user Registration/Login)
2. Files (Includes pdf, images, text)
3. Biometrics (Image)

2.2 Tools:

The following tools are used to build the project:

Python, Flask, Infura, Redis, TensorFlow, PyTorch, pycryptodome, OpenCV, WTforms, jinja template, pymongo, GridFS, Threading Module, HTML, CSS, JavaScript, Time and OS Modules.

2.3 Procedure:

1. The user first registers himself on the website (if he's accessing it for the first time) or logs in to the website (if he's already registered).
2. He then records/registers all the recipients with whom he is sharing the file.
3. He can add or remove the receivers by searching for their username.
4. The file is then uploaded to the IPFS server by clicking on the upload button.
5. The user can view his past uploads and the current status of the file, as well as check the files he has received from other users.
6. The receiver can access the uploaded file by using the CID (Content Identifier) sent by the sender.
7. The receiver now has two ways to send the file:
 - He can do a normal file transfer where the file is encrypted and sent back to the sender for further corrections and modifications.
 - Or, he can digitally sign the document and send it back to the sender.
8. The receiver can send the required corrections by using the comment box provided on the website.
9. The process is complete only when the file has been digitally signed.

2.4 Output:

1. The file is uploaded to the IPFS server.
2. The receiver can download the file.
3. The receiver either send the document if corrections are needed, or digitally sign the document as a sign of approval.

CHAPTER 3

REQUIREMENT SPECIFICATION

3.1 Functional Requirements:

The system should have the following functional requirements to effectively fulfil its purpose:

1. **User Interface:** The system should provide a user-friendly website interface for seamless communication with users, facilitating efficient document uploading.
2. **Decentralized Storage:** The system should utilize an IPFS server to store and share documents, leveraging the advantages of decentralization.
3. **Encryption and Decryption:** The system should support encryption and decryption mechanisms (such as RSA or AES) to ensure the confidentiality and security of the documents.
4. **Document Upload:** Authorized parties should be able to upload documents to the IPFS through the system.
5. **Document Sharing:** The system should enable authorized parties to securely share documents with other authorized parties.
6. **Document Download:** Authorized parties should have the capability to download documents from the IPFS through the system.
7. **Document Verification:** The system should allow authorized parties to verify the authenticity and integrity of documents using the cryptographic hash stored on the IPFS.
8. **Access Control:** The system should incorporate access control mechanisms to restrict document access and modification only to authorized parties.
9. **Corrections:** The system should facilitate the process of sending required corrections back to the sender, ensuring effective collaboration and document accuracy.
10. **Digital Signing:** The system should provide a feature for authorized users to digitally sign documents, enhancing their authenticity and ensuring non-repudiation.
11. **Timestamping:** The system should include a timestamping mechanism that records the date and time of each document exchange on the IPFS, establishing a transparent and secure transaction history.

12. User Management: The system should offer user management functionalities, allowing authorized parties to manage their accounts, credentials, and permissions within the system.

Overall, the functional requirements for a digital exchange of documents using IPFS should ensure that the system provides a secure, decentralized means to exchange and verify the documents.

3.2 Non-Functional Requirements:

Non-functional requirements are those requirements that describe the performance, quality, and operational characteristics of the system. Some key non-functional requirements that should be considered include:

1. Performance: The system should be designed to handle a high volume of document exchanges and transactions with minimal latency and response times. It should also be able to handle concurrent transactions and be scalable to accommodate future growth.
2. Reliability: The system should be reliable and available at all times, with minimal downtime or service interruptions. It should also provide mechanisms for backup and recovery in the event of system failure or data loss.
3. Security: The system should be secure and resistant to attacks, with robust authentication, authorization, and encryption mechanisms to protect the confidentiality and integrity of the exchanged documents.
4. Interoperability: The system should be interoperable with other systems and applications, enabling seamless integration with existing processes and workflows.
5. Usability: The system should be user-friendly and intuitive, with a simple and intuitive user interface that allows users to easily upload, download, and share documents. It should also support a range of document formats and sizes.
6. Compliance: The system should comply with legal and regulatory requirements, such as data protection and privacy laws. It should also provide mechanisms for auditing and reporting to support compliance and accountability.
7. Scalability: The system should be scalable to meet the needs of different users and use cases, with the ability to handle large volumes of transactions and support a wide range of document types.

3.3 Software Requirements:

The Software requirement Specification is the official statement of what is required of the system developers. The requirement set out in this document is complete and consistent.

Software required to develop a project:

- Operating System: Windows/MacOS/Linux.
- Tool/IDE: Microsoft Visual Studio, GitBash, GitHub.
- Front End: HTML, CSS, JavaScript, Jinja.
- Back End: Python, Flask.
- IPFS Client: Infura API.
- Biometric authentication library: OpenCV, FaceNet.
- Digital signature library: OpenSSL.
- Web development framework: Flask.
- Database: pymongo.

3.4 Hardware Requirements:

Hardware required to develop the project:

- Processor: Intel i5 or above
- RAM: 8GB and above.
- Hard Disk: 5 GB and Above.

Hardware required to run the application:

- Any type of OS with Internet Access.

CHAPTER 4

DETAILED DESIGN

4.1 Overview:

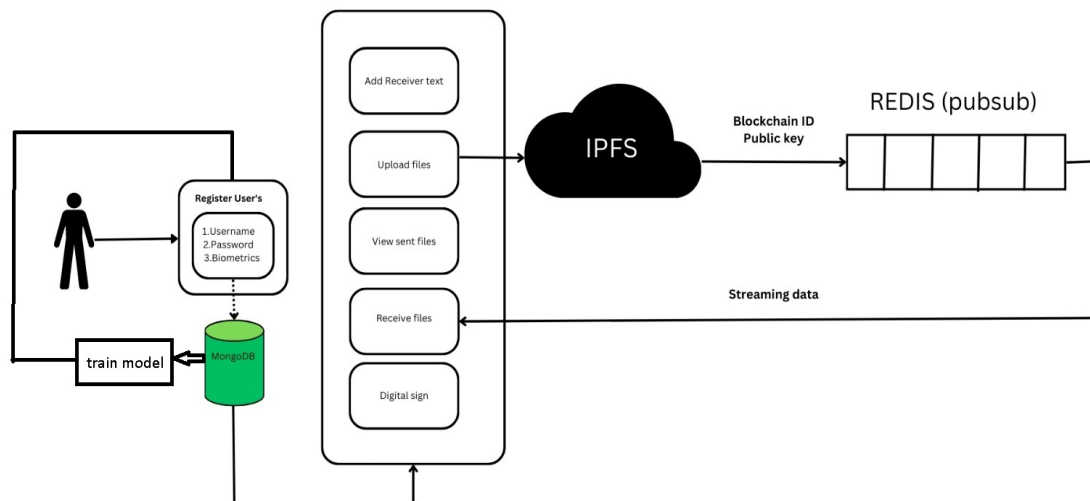


Fig 4.1 Overview of the System

Fig 4.1 gives a general overview of the system and the major components used:

1. First step is for users is to register in the website, this will redirect the user to collect their images which are then uploaded to mongodb.
2. On selecting face recognition as the mode for login, the images from mongo are downloaded.
3. A model is trained which is used in front-end for face recognition
4. On logging in we have five main views or functionality
 - Uploading files to IPFS and publish its information through REDIS.
 - Add receivers to accept the files from the sender.
 - View all the files that have been uploaded.
 - View all the files which u have received from others this will be a listener to REDIS.
 - Digitally sign and approve the documents sent to you.

4.2 Use case Diagram:

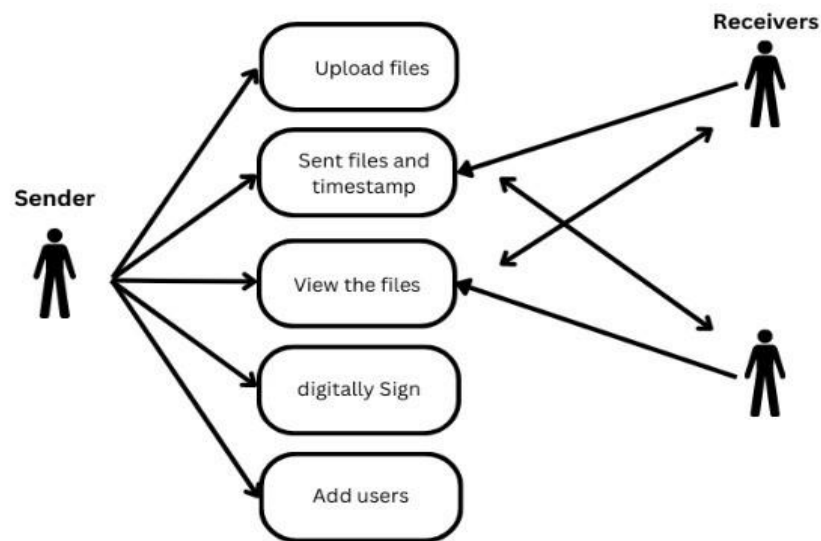


Fig 4.2 Use case Diagram

The above use case diagram in fig 4.2 shows how the users will interact with our project and its five main functionalities.

1. There are two main kinds of users
 - Those who send files and
 - Those who receive those files
2. The sender can view and perform all the activities but a receiver can only view files and react to that by either,
 - Digitally signing those files, or
 - By Adding comments in the box provided (in the website)

4.3 Sequence Diagram:

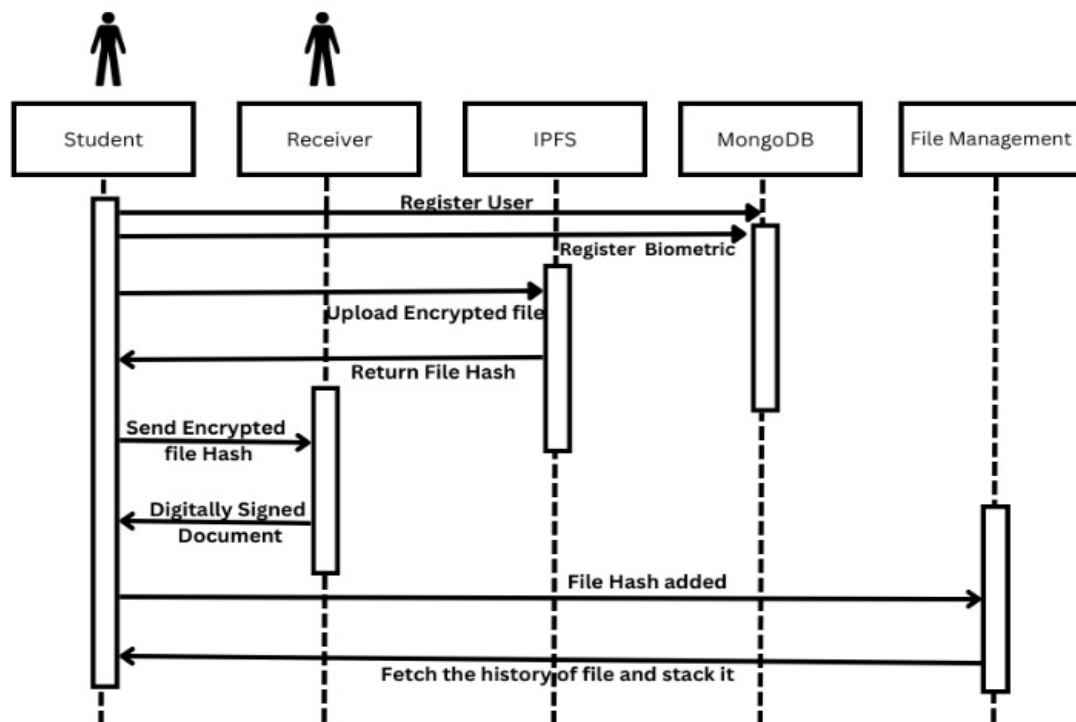


Fig 4.3 Sequence diagram

Fig 4.3 shows the sequence diagram that shows the interactions between objects or components in a system over time.

1. The user should register himself by providing his personal details such as email Id, username and password.
2. After providing the details he will be directed to the next page where he will register his face, which will be used to authenticate the user and the generate private key .
3. Once the private key is generated the user has to download the private key and keep it securely, as private is generated only once.
4. After registering the user can login and use all the services.
5. The user can upload the file and send to multiple users and in order to do so he has to provide his private key.
6. The file uploaded will be encrypted and will be uploaded to the ipf s server and the file hash (CID) of that file will be sent to the receiver.

7. The receiver can view the file file , download it and then sign the document if it is correct using his private key .
8. Once the document is signed it will be reflected to the sender, in the website.
9. The sender can verify the authenticity of that file.

CHAPTER 5

IMPLEMENTATION

5.1 Programming Language:

Python: Python is a popular programming language that was created by Guido van Rossum and introduced in 1991. It is widely used due to its high-level nature and interpreted execution, making it accessible for beginners. Python is an open-source language, meaning it is freely available for use, distribution, and modification. Its versatility extends to various domains, including web development, scientific computing, data analysis, artificial intelligence, and machine learning.

Python stands out for its clear and readable syntax, making it easy to understand and learn. This simplicity contributes to its reputation as an ideal language for beginners. The language has a thriving community of developers who actively contribute to its growth by creating libraries and tools that enhance its functionality and ease of use. These resources enable users to perform a wide range of tasks efficiently.

Python supports multiple programming paradigms, including object-oriented, functional, and procedural programming, accommodating diverse coding styles. Additionally, Python boasts a comprehensive standard library, offering a vast collection of pre-built functions and modules. These resources facilitate tasks such as data manipulation, network communication, web development, and much more.

One of Python's notable strengths is its emphasis on readability and maintainability. This quality ensures that code written in Python is easy to comprehend and modify, promoting collaboration and teamwork among developers. Its versatility, readability, and extensive ecosystem of libraries and tools have established Python as a powerful and favoured programming language, utilized extensively across various industries and academic fields. Whether for beginners or experienced programmers, Python provides a robust foundation for building diverse applications.

5.2 Frameworks:

Flask: Flask is a Python web framework known for its lightweight and flexible nature. Created by Armin Ronacher in 2010, Flask enables developers to rapidly build web applications without the complexities typically associated with full-stack frameworks.

One of the key features of Flask is its extensibility. It is built on top of the Werkzeug WSGI toolkit and the Jinja2 templating engine, allowing developers to easily incorporate third-party extensions for additional functionality. A wide range of extensions are available for Flask, covering areas such as database integration, authentication, testing, and more.

Flask's minimalistic approach is another advantage. By focusing on core functionalities and avoiding unnecessary components, Flask offers developers greater control over their application's architecture. This aspect proves beneficial for small to medium-sized projects, where a lightweight framework like Flask can provide faster and more efficient performance compared to larger frameworks.

Simplicity and user-friendliness are distinguishing qualities of Flask. Its gentle learning curve makes it particularly suitable for beginners venturing into web development. The framework boasts well-structured documentation complemented by numerous examples and tutorials accessible online.

In summary, Flask is a powerful and versatile Python web framework. Its minimalistic design, extensibility, and beginner-friendly nature contribute to its popularity among developers seeking a flexible and efficient solution for web application development.

5.3 Algorithms:

We make use of the following algorithms.

- **RSA** - The RSA (Rivest-Shamir-Adleman) algorithm is a popular asymmetric encryption algorithm widely used in cryptography. Named after its creators, the RSA algorithm leverages the mathematical properties of prime numbers and modular arithmetic. Its primary applications include secure data transmission, digital signatures, and key exchange protocols. In RSA, participants generate a pair of cryptographic keys: a public key for encryption and a private key for decryption. The security of RSA rests on the difficulty of factoring large prime numbers, as the keys are derived from the product of two such primes.

There are several reasons for the widespread use of the RSA algorithm. Firstly, it offers a high level of security by capitalizing on the challenge of factoring large numbers. The algorithm's security relies on the computational complexity of factoring, making it suitable for safeguarding sensitive information and ensuring secure communication. Secondly, RSA supports digital signatures, which are crucial for verifying the authenticity and integrity of digital documents. By signing a document with their private key, the sender allows the recipient to use the corresponding public key to verify the signature and ensure the document's integrity. Lastly, RSA's extensive adoption is attributed to its established and standardized nature. It is supported by various cryptographic libraries and protocols, facilitating its implementation across different systems and platforms. This wide acceptance guarantees interoperability and compatibility, making RSA a dependable choice for secure communication and cryptographic operations.

In summary, the RSA algorithm is valued for its robust security properties, support for digital signatures, and broad adoption in cryptographic systems. Its ability to ensure secure data transmission, authentication, and integrity verification makes it suitable for diverse applications requiring strong cryptography.

- **Fernet (AES)** - Fernet is a widely used symmetric encryption algorithm that employs the AES (Advanced Encryption Standard) algorithm in CBC (Cipher Block Chaining) mode with PKCS7 padding. It is known for its simplicity, security, and efficiency in cryptography. Fernet is commonly utilized for securing message encryption and decryption, ensuring the confidentiality and integrity of data. The algorithm relies on a shared secret key that is used by both the sender and the receiver for encryption and decryption operations, guaranteeing that only authorized parties can access and decrypt the encrypted data.

The decision to use Fernet (AES) is driven by its strong security features. AES is a trusted encryption algorithm that has been adopted as the standard for symmetric encryption by organizations like the U.S. National Institute of Standards and Technology (NIST). With AES employing a block cipher and offering various key sizes, it is highly resistant to brute-force attacks. Fernet enhances the security by utilizing AES in CBC mode, which introduces an additional layer of protection by leveraging the ciphertext of the previous block during encryption. This approach mitigates the risk of identical plaintext blocks producing identical ciphertext blocks, thereby bolstering security and preventing potential vulnerabilities.

Efficiency is another notable advantage of using Fernet (AES). AES is known for its speed and efficiency in encrypting large volumes of data promptly. Fernet's implementation of AES in CBC mode ensures both security and optimized performance. The adoption of PKCS7 padding enables the encryption and decryption of plaintext with arbitrary lengths, ensuring accurate retrieval of the data. Furthermore, Fernet (AES) enjoys broad support and implementation across various programming languages and cryptographic libraries, making it highly accessible and compatible.

In summary, Fernet (AES) is favoured for its robust security, efficiency, and widespread adoption. By utilizing the AES algorithm in CBC mode with PKCS7 padding, Fernet provides a reliable and secure symmetric encryption solution. It effectively safeguards data confidentiality and integrity, making it well-suited for diverse applications where secure message encryption and decryption are essential.

- **Backpropagation** - Backpropagation is a widely utilized optimization technique employed in artificial neural networks, playing a crucial role in training deep learning models. Its primary objective is to adjust the neural network's weights and biases through gradient-based optimization. The aim is to minimize the disparity between the predicted output and the actual output when the network is presented with training data.

In the domain of face recognition, backpropagation holds significant importance in training deep neural networks for tasks such as face detection, identification, and verification. These networks undergo training on a substantial dataset comprising labelled face images, enabling them to learn and extract relevant features from input images and make precise predictions. The backpropagation algorithm computes the gradients of the network's parameters concerning a selected loss function. This enables the network to update its weights and biases, progressively minimizing the loss.

Throughout the training process, backpropagation calculates the error between the predicted outputs and the ground truth labels. It then propagates this error backward through the network's layers, computing the parameter gradients. These gradients are subsequently utilized to update the weights and biases using an optimization algorithm like stochastic gradient descent (SGD). This iterative process continues until the network reaches a state where the loss is minimized.

By employing backpropagation, face recognition models can effectively learn intricate patterns and features present in face images, leading to enhanced accuracy and robustness. The ability to fine-tune deep neural networks' parameters through backpropagation allows for the capture of nuanced details and subtle variations in faces, thereby facilitating more accurate classification and identification of individuals.

- **Elliptical Curve Algorithm** - The elliptic curve algorithm (ECA) is a widely used public-key cryptography algorithm that ensures secure communication over networks. It leverages the mathematical properties of elliptic curves and finds applications in digital signature schemes, key exchange protocols, and various cryptographic scenarios. ECA's core concept involves creating a public key that can be shared openly and a private key that remains confidential. The security of the system is based on the immense difficulty of deriving the private key from the public key, even with advanced computing capabilities.

In the context of this project, the elliptic curve algorithm can be employed for secure key generation, key exchange, and digital signatures. By utilizing the algorithm's properties, cryptographic operations like key generation and exchange can establish secure encryption keys, safeguarding them against compromise. Furthermore, the algorithm enables efficient and secure digital signatures, ensuring the integrity and authenticity of signed documents.

To generate a public key, the owner selects a random integer as the private key and performs a multiplication operation with a base point on the curve. The resulting point becomes the public key. When encrypting a message, the sender employs the recipient's public key to derive a shared secret key for encryption. The recipient can then use their private key to derive the same shared secret key and decrypt the message. Signing a message involves using the sender's private key to generate a digital signature, which is attached to the message. The recipient can subsequently use the sender's public key to verify the signature and confirm the message's integrity.

One notable advantage of the elliptic curve algorithm is its efficiency. It necessitates shorter key lengths compared to other algorithms while maintaining equivalent or even stronger security levels.

5.3.1 Why Elliptical Curve Algorithm?

There are several reasons why elliptic curve cryptography (ECC) is sometimes preferred over RSA cryptography:

1. **Key size:** ECC provides the same level of security as RSA with a smaller key size [30]. For example, a 256-bit ECC key is considered to be as secure as a 3072-bit RSA key. This smaller key size results in faster computations, which is important for applications that require real-time or near real-time performance.

Key Size Comparison:

Symmetric Key Size (bits)	RSA Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Fig 5.1: Key Size Comparison between Symmetric Key, RSA, ECC

2. **Speed:** ECC is faster than RSA when it comes to key generation, encryption, and decryption operations. This makes it more suitable for resource-constrained environments like mobile devices and embedded systems.
3. **Memory requirements:** ECC requires less memory compared to RSA for storing keys and performing cryptographic operations. This is particularly important for devices with limited memory resources.
4. **Resistance to quantum computing attacks:** ECC is believed to be more resistant to attacks by quantum computers than RSA. This is because the algorithms used in ECC are based on the mathematical properties of elliptic curves, which are believed to be more resistant to quantum computing attacks.

5. Industry acceptance: ECC is becoming increasingly accepted as a standard for encryption and digital signatures in industry and government sectors. This means that there is a growing ecosystem of hardware and software products that support ECC, making it easier to adopt and integrate into existing systems.

Overall, the use of the elliptic curve algorithm offers a combination of strong security, computational efficiency, shorter key lengths, and wide acceptance. These factors make it an attractive choice for applications requiring secure communication, digital signatures, and key exchange, particularly in resource-constrained environments or scenarios where efficiency and speed are crucial.

CHAPTER 6

TESTING

6.1 Registration Page Testing:

1. **Functionality Testing:** Verify that all registration page features work correctly, including form validation, error handling, and displaying confirmation messages.
2. **Usability Testing:** Evaluate the ease of use of the registration page by assessing how intuitive the form is to complete, the clarity of instructions, and whether users can successfully register without assistance.
3. **Performance Testing:** Measure the speed and responsiveness of the registration page, including page load times and the time it takes for users to complete the registration process.
4. **Compatibility Testing:** Ensure the registration page functions properly across different browsers, devices, and operating systems, verifying its compatibility and responsiveness.
5. **Security Testing:** Assess the security measures of the registration page, including data protection, prevention of unauthorized access, and encryption of sensitive information.
6. **Accessibility Testing:** Check the accessibility of the registration page for users with disabilities, confirming compatibility with assistive technologies and adherence to web accessibility standards.
7. **A/B Testing:** Compare the effectiveness of different registration page designs or features by analyzing user responses and behaviors to different versions of the page.

6.2 Face Recognition Testing:

1. **Accuracy Testing:** Test the accuracy of the face recognition system by measuring how well it can correctly identify individuals based on their facial features. This can be done by using a dataset of known faces and comparing the system's results against those of human experts.
2. **Performance Testing:** Test the performance of the face recognition system by measuring how quickly it can process images and identify individuals. This can be done by measuring the time it takes the system to process a set of images.

3. **Robustness Testing:** Test the robustness of the face recognition system by measuring how well it can handle variations in lighting, facial expressions, and other factors that may affect the accuracy of the system.

6.3 File Upload Testing:

1. **File Type Validation:** Test whether the system accepts only the specified file types and rejects other file types.
2. **File Size Validation:** Test whether the system accepts only the specified file sizes and rejects larger files.
3. **Concurrent Uploads:** Test whether the system allows multiple users to upload files at the same time and whether it can handle concurrent uploads without any issues.
4. **Error Handling:** Test whether the system displays appropriate error messages when there is an issue with the file upload, such as a file size limit or an incorrect file type.
5. **Security Testing:** Test whether the system is secure and does not allow malicious files to be uploaded.
6. **Performance Testing:** Test whether the file upload process is fast and responsive, and can handle large files without any issues.
7. **Compatibility Testing:** Test whether the file upload functionality works properly on different browsers and devices.
8. **Usability Testing:** Test whether the file upload process is intuitive and easy to use for users, and whether there is sufficient guidance provided during the upload process.

6.4 Digital Signature Testing:

1. **Validation Testing:** Test the digital signature system's ability to correctly validate digital signatures and ensure that they have not been tampered with. This can be done by verifying the signatures on a set of known documents and comparing the results with the expected outcomes.
2. **Integrity Testing:** Test the digital signature system's ability to ensure the integrity of signed documents, by intentionally modifying a signed document and verifying whether the digital signature is still valid.

3. **Compatibility Testing:** Test the digital signature system's compatibility with different file formats and software platforms, including web browsers, document management systems, and other relevant applications.
4. **Performance Testing:** Test the digital signature system's performance, including the time it takes to generate and verify digital signatures, the system's response times, and the system's ability to handle large volumes of signatures.
5. **Security Testing:** Test the security of the digital signature system, including whether the system uses appropriate encryption and hashing algorithms to protect signatures, and whether it can detect and prevent attacks such as replay attacks, man-in-the-middle attacks, and other forms of forgery.
6. **Usability Testing:** Test the usability of the digital signature system, including how easy it is for users to sign and verify documents, and whether the system provides appropriate guidance and feedback during the signing processes.
7. **Compliance Testing:** Test the digital signature system's compliance with relevant regulations and standards, including eIDAS, UETA, ESIGN, and other relevant regulations.

6.5 File Encryption and Decryption Testing:

1. **Encryption/Decryption Accuracy Testing:** Verify the accuracy of the encryption and decryption system by encrypting a set of known files and comparing the results with the expected outcomes. Similarly, decrypt the encrypted files and ensure the original files are correctly recovered.
2. **Key Management Testing:** Evaluate the system's ability to securely manage encryption keys. Test key generation, storage, and sharing mechanisms to ensure the keys are properly handled and protected throughout their lifecycle.
3. **Performance Testing:** Measure the performance of the encryption and decryption system. Assess factors such as encryption and decryption speeds, system response times, and its ability to handle large volumes of data efficiently.
4. **Compatibility Testing:** Test the compatibility of the encryption and decryption system with different file formats and software platforms. Ensure seamless functionality across web browsers, document management systems, and other relevant applications.

5. **Security Testing:** Assess the security measures implemented in the encryption and decryption system. Verify if the system utilizes appropriate encryption algorithms and key lengths to protect files. Additionally, test for vulnerabilities against attacks like brute force attacks, side-channel attacks, and other hacking attempts.
6. **Usability Testing:** Evaluate the usability of the encryption and decryption system. Focus on how user-friendly it is for individuals to encrypt and decrypt files. Determine if the system provides clear instructions, guidance, and feedback throughout the process.
7. **Compliance Testing:** Verify if the encryption and decryption system complies with relevant regulations and standards such as HIPAA, GDPR, PCI-DSS, and other applicable requirements.

CHAPTER 7

EXPERIMENTAL RESULTS

7.1 Code Pack:

In this section we discuss about the important modules and functions used in the project:

1. File Upload:

```
def file_upload():
    form = UploadFileForm()
    if request.method == 'POST':
        # username = request.form['options[]']
        username = request.form.getlist('options[]')
        login_user = db1.userdata.find_one({'name': session['username']})

        file = request.files['file'] # First grab the file
        file.save(os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'], secure_filename(file.filename))) # Then save the file
        directory = os.path.join(os.path.abspath(os.path.dirname(__file__)), app.config['UPLOAD_FOLDER'])
        files = os.listdir(directory)
        files.sort(key=lambda x: os.path.getctime(os.path.join(directory, x)), reverse=True)
        recent_file = files[0]
        print("this is ", recent_file)
        with open(dir_name + recent_file, "rb") as f:
            data = f.read()
        encrypted_file = fernet.encrypt(data)
        with open(dir_name + recent_file, "wb") as f:
            f.write(encrypted_file)
        for f in files:
            files.sort(key=lambda x: os.path.getctime(os.path.join(directory, x)), reverse=True)
            recent_file = files[0]
            item = open(dir_name + recent_file, 'rb')
            items[f] = item
        response = requests.post("https://ipfs.infura.io:5001/api/v0/add?pin=true&wrap-with-directory=false",
                                auth=(proj_id, proj_secret), files=items)
        # print(result)
        dec = json.JSONDecoder()

        while i < 1:
            data, s = dec.raw_decode(response.text[i:])
            i += s + 1
            if data['Name'] == '':
                data['Name'] = 'Folder CID'
            print("%s: %s" % (data['Name'], data['Hash']))
            x = data['Hash']
            # db.filedata.delete_many({})
            if login_user:
                for i in username:
                    db.filedata.insert_one({"from": session['username'], "to": i, "filehash": x, "filename": file.filename, "signature": "none", "status": "none", "feedback": "none"})

            # https://VASDoc.infura-ipfs.io/ipfs/

        # return "<h2>Click this link{</h2>".format(x)
        # https://VASDoc.infura-ipfs.io/ipfs/

        gateway="https://VASDoc.infura-ipfs.io/ipfs/"
        print(requests.get(url=gateway+data['Hash']).text)
        data = requests.get(url=gateway+data['Hash']).text
        decrypted_file = fernet.decrypt(data)
        with open("dec.pdf", "wb") as f:
            f.write(decrypted_file)

        # return redirect("https://VASDoc.infura-ipfs.io/ipfs/"+x)
        return redirect(url_for("Profile.profile"))
    return render_template('file_upload.html', form=form)
```

Fig 7.1: Code snippet of file upload function

The above code snippet is the part of a web application that handles file uploads and performs various operations on the uploaded files using IPFS and encryption. Here's a breakdown of the code:

1. It starts by creating an instance of an **UploadFileForm**, which likely represents an HTML form that allows users to select and upload files.
2. The code checks if the request method is **POST**, indicating that a file has been submitted via the form.
3. It retrieves the selected username(s) from the form. The exact implementation of the **UploadFileForm** class and the form structure is not shown in the provided code.
4. It retrieves the uploaded file from the request using **request.files['file']** and saves it to a specified directory using **file.save()**.
5. The code then retrieves the most recently uploaded file from the directory by sorting the files based on their creation time.
6. It reads the contents of the file, encrypts the data using the fernet encryption scheme, and overwrites the original file with the encrypted data.
7. Next, it iterates over the files in the directory, sorting them based on creation time, and creates a dictionary of open file objects.
8. It makes a **POST** request to an IPFS node (in this case, the **Infura IPFS API**) to add the files to IPFS. The files are sent as part of the request using the **files** parameter. The response from IPFS is stored in the response variable.
9. The code parses the response JSON to **retrieve the CID** (Content Identifier) of the uploaded file.
10. If a user is logged in (**login_user** exists), it iterates over the selected usernames and inserts a document into the **filedata** collection in a MongoDB database. This document contains information such as the sender, recipient, file hash, filename, signature status, and feedback.
11. It constructs a URL to the IPFS gateway and retrieves the data of the uploaded file from the gateway using the CID.
12. It decrypts the retrieved data using the **fernet decryption scheme** and writes it to a file named "dec.pdf".
13. Finally, it redirects the user to the **Profile.profile** route and displays the profile page.

2. Public and Private key generation:

```
def genKeys():
    j = get_img(session['username'])

    print(j)
    val = 0
    for i in range(0,len(j)):
        if type(i) is int:
            val += i

    print(val)

    privKey = secrets.randbelow(10**14)

    privKey = privKey * 116 * val
    print("Private key :",privKey)
    with open("private.txt","w") as f:
        f.write(str(privKey))

    pubKey = (generator_secp256k1 * privKey).pair()
    print(pubKey)
    pubKey = str(pubKey)
    db.userdata.find_one_and_update({'name': session['username']}, {"$set":{"pubKey":pubKey}})

    return render_template("file_download.html")
```

Fig 7.2: Code snippet of generate keys function

The above code snippet is used to generate a pair of public and private keys for a user and store the private key in a file. Here's the explanation of the code:

1. It starts by calling the **get_img()** function with the **username** stored in the user's session. The purpose and implementation of this function are not provided in the code snippet.
2. The code initializes a variable **val** to 0.
3. It iterates over the elements of the **j** list, which is presumably obtained from the **get_img()** function. For each element, it checks if the type is an integer (**type(i) is int**) and if true, adds the integer value to **val**.
4. It generates a random private key using the **secrets.randbelow()** function. The generated private key is limited to a value less than 10^{14} .
5. The private key is modified by multiplying it with **116** and **val**. This operation seems to introduce additional randomness and uniqueness to the private key based on the computed **val**.
6. The private key is written to a file named "private.txt" using the **with open("private.txt","w") as f** statement.

7. The public key is calculated by multiplying the **generator_secp256k1** (presumably a predefined generator point on the elliptic curve) with the private key. The resulting public key is represented as a pair.
8. The public key is converted to a string representation using **str(pubKey)**.
9. The function updates the **pubKey** field of the user's data in the database using the **db.userdata.find_one_and_update()** method. The **pubKey** value is set to the generated public key string.
10. Finally, the function renders a template named "file_download.html". The purpose and content of this template are not provided in the code snippet.

3. Encryption of file using RSA Algorithm:

```
def encrypt_data(msg,public_key):
    k = b""
    k = public_key
    encrypted_msg = rsa.encrypt(msg.encode(),k)
    with open("encrypted.txt" , "wb") as f:
        f.write(encrypted_msg)
    return encrypted_msg
```

Fig 7.3 Code snippet of file encryption using RSA Algorithm

This code defines a function named **encrypt_data()** that encrypts a given message using the RSA algorithm and a provided public key. Here's an explanation of the code:

1. The function takes two parameters: **msg** (the message to be encrypted) and **public_key** (the RSA public key).
2. It initializes a variable **k** as an empty byte string (**b""**).
3. The variable **k** is assigned the value of the **public_key**. It seems that the **public_key** is expected to be a byte string representation of an RSA public key.
4. The message **msg** is encrypted using the **rsa.encrypt()** function from a module/library called **rsa**. The **msg** is first encoded to bytes using **.encode()** method before being passed to the **rsa.encrypt()** function along with the **public_key**. The result is stored in the variable **encrypted_msg**.
5. The encrypted message is written to a file named "encrypted.txt" using the **with open("encrypted.txt" , "wb") as f** statement. The **wb** mode ensures that the file is opened in binary mode for writing.

- Finally, the encrypted message (**encrypted_msg**) is returned from the function.

4. AES Encryption:

```

from Crypto.Random import get_random_bytes
from Crypto.Protocol.KDF import PBKDF2
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad , unpad

simple_key = get_random_bytes(32)
salt = simple_key
password = "kk123"
key = PBKDF2(password,salt,dkLen=32)
print(key)
with open("1BY19CS109_FEES.pdf","rb") as f:
    msg = f.read()

cipher = AES.new(key, AES.MODE_CBC)
print(cipher)
ciphered_data = cipher.encrypt(pad(msg,AES.block_size))
# print(ciphered_data)

with open("encrypted.bin" ,"wb") as f :
    f.write(cipher.iv) # Creates a binary fog
    f.write(ciphered_data)

with open("encrypted.bin","rb") as f:
    iv = f.read(16)
    decrypt_data = f.read()

cipher = AES.new(key,AES.MODE_CBC,iv = iv)
original = unpad(cipher.decrypt(decrypt_data),AES.block_size)

with open("decrypted.pdf","wb") as f:
    f.write(original)

```

Fig 7.4: Code snippet of AES encryption

This code snippet demonstrates the encryption and decryption of a file using the AES (Advanced Encryption Standard) algorithm. Here's an explanation of the code:

- The code imports necessary modules from the Crypto library, including **get_random_bytes** for generating random bytes, **PBKDF2** for key derivation, **AES** for the cipher, and **pad** and **unpad** for data padding.
- A random key of 32 bytes (**simple_key**) is generated using **get_random_bytes(32)**.

3. The `simple_key` is used as the salt for key derivation (PBKDF2). A password (**password**) is provided, and the key is derived with a length of 32 bytes (`dkLen=32`).
4. The code opens a file named "1BY19CS109_FEES.pdf" in read binary mode and reads its contents into the **msg variable**.
5. An **AES cipher object** is created (`cipher = AES.new(key, AES.MODE_CBC)`) with the derived key (`key`) and the AES mode set to Cipher Block Chaining.
6. The message (`msg`) is padded using `pad()` to ensure it aligns with the block size of AES. The resulting padded data is then encrypted using the cipher object (`cipher.encrypt()`), and the encrypted data is stored in **ciphered_data**.
7. The code opens a file named "**encrypted.bin**" in write binary mode and writes the initialization vector (**`cipher.iv`**) followed by the encrypted data (**`ciphered_data`**) into the file.
8. The code opens the "**encrypted.bin**" file in read binary mode, reads the first 16 bytes as the initialization vector (`iv`), and reads the remaining data as the encrypted message (`decrypt_data`).
9. A new **AES cipher object** is created (`cipher = AES.new(key, AES.MODE_CBC, iv=iv)`) with the derived key (`key`), AES mode set to **CBC**, and the initialization vector (`iv`).
10. The encrypted message (`decrypt_data`) is decrypted using the cipher object (`cipher.decrypt()`), and the result is unpadded using `unpad()` to remove any padding added during encryption. The original message is stored in the `original` variable.
11. The code opens a file named "**decrypted.pdf**" in write binary mode and writes the original message (`original`) into the file.

5. File Approval:

```
def approve():
    fileHash = request.args.get('fileHash')
    splitted= fileHash.split(":")
    fileHash = splitted[0]
    filename = splitted[1]
    extension = filename.split(".")
    gateway="https://VASDoc.infura-ipfs.io/ipfs/"
    data = requests.get(url=gateway+fileHash).text
    decrypted_file = fernet.decrypt(data)

    with open(os.getcwd()+r"\\dec_downloads\\"+"dec"+datetime.datetime.now().strftime("%Y-%m-%d-%S")+extension[1], 'wb') as f:
        f.write(decrypted_file)
    if request.method == 'POST':
        privKey = request.form['privateKey']
        privKey = int(privKey)
        print(type(privKey))
        comment = request.form['comment']
        status = request.form['Approve']
        print(status)
        # msg = 'Qmc3VKBDzu5LuVu6sW7AWpBdMwU4imR6oCxWFBverWdcXw'
        msg = fileHash
        # privKey = 10723831103338713018195600
        if status == "Approve":
            msg = fileHash
            signature = sign(msg,privKey,generator_secp256k1,hashlib,sign)
            found = db.filedata.find({"filehash":fileHash})
            approved = db1.approve_file.find_one({"filehash":fileHash})
            signature = str(signature)

            if found and not approved :
                for f in found:
                    received_from = f["from"]
                    db.filedata.find_one_and_update({"filehash":fileHash},{"$set":{"signature":signature,"status":"approved","feedback":comment}})
                    # db1.approve_file.insert_one({"filehash":fileHash,"filename":filename,"received_from":received_from,"to": session["username"],"signature":signature,"status":status})
                return redirect(url_for("Profile.profile"))
            else:
                found = db.filedata.find({"filehash":fileHash})
                approved = db1.approve_file.find_one({"filehash":fileHash})
                if found and not approved:
                    for f in found:
                        received_from = f["from"]
                        db.filedata.find_one_and_update({"filehash":fileHash},{"$set":{"signature":signature,"status":"unapproved","feedback":comment}})
                        # db1.approve_file.insert_one({"filehash":fileHash,"filename":filename,"received_from":received_from,"to": session["username"],"status":"UnApproved"})
                return redirect(url_for("Profile.profile"))
```

Fig 7.5: Code snippet of file approval function

The above is the code snippet which defines the file approve function. Here's an explanation of the code:

1. The function is triggered when a request is made to the **/approve** endpoint.
2. The function retrieves the **fileHash** parameter from the request's query string and splits it into fileHash and filename variables. It also extracts the file extension from the filename.
3. It constructs the IPFS gateway URL (**gateway**) and makes a request to retrieve the file's data from the IPFS network using the provided fileHash. The retrieved data is stored in the data variable.
4. The encrypted file is decrypted using **fernet.decrypt()** with the retrieved data, and the decrypted file is saved to a file with a unique name in the **dec_downloads** directory.

5. If the request method is POST, the function retrieves the **privateKey**, comment, and status values from the form data. The privateKey is **converted** to an integer.
6. If the status is "**Approve**" the function proceeds with digital signature generation and verification. It generates a signature for the fileHash using the `signm()` function, the `privKey`, `generator_secp256k1`, `hashlib`, and `sign` objects. The signature is converted to a string.
7. The function checks if the file with the given **fileHash** exists in the `filedata` collection and if it has not been previously approved. If so, it updates the corresponding document in the `filedata` collection with the generated signature, status, and feedback. Alternatively, if the status is **not "Approve"** it updates the document with "none" for the signature and "unapproved" for the status.
8. Finally, the function redirects the user to the **Profile.profile** endpoint.

6. File Decryption:

```
def decrypt_data(directory):
    files = os.listdir(directory)
    files.sort(key=lambda x: os.path.getctime(os.path.join(directory, x)), reverse=True)
    recent_file1 = files[0]
    recent_file2 = files[1]
    print("recent 1 :", recent_file1)
    print("recent 2 :", recent_file2)
    file_extension = os.path.splitext(recent_file1)[1]
    print(file_extension)

    if file_extension == ".txt":
        pass
    else :
        temp = recent_file2
        recent_file2 = recent_file1
        recent_file1 = temp

    with open(dir_name + recent_file1, "rb") as f:
        msg = f.read()
        # print(msg)

    with open(dir_name + recent_file2, "rb") as f:
        private = rsa.PrivateKey.load_pkcs1(f.read())

    decrypted_msg = rsa.decrypt(msg, private)
    return decrypted_msg.decode()
```

Fig 7.6: Code snippet of file decrypt function

The `decrypt_data` function appears to be a part of a larger codebase and is responsible for decrypting data using RSA encryption. Here's an explanation of the code:

1. The function takes a **directory parameter**, which represents the directory containing the encrypted files.
2. The function retrieves the list of files in the directory using **os.listdir()** and sorts them based on the creation time in descending order.
3. It selects the two most recent files from the sorted list. The variable `recent_file1` represents the most recent file, and `recent_file2` represents the second most recent file.
4. The function determines the file extension of `recent_file1` using **os.path.splitext()** and stores it in the **file_extension** variable.
5. If the file extension is ".txt", the function does nothing. This suggests that the function might have different logic for handling text files compared to other file types.
6. If the file extension is not ".txt", the function swaps the values of `recent_file1` and `recent_file2`. This suggests that the order of the files might be important for decryption in this case.
7. The function reads the contents of `recent_file1` as binary data using `open()` and `rb` mode.
8. It reads the contents of `recent_file2` as a **private key** using the **rsa.PrivateKey.load_pkcs1()** function. This suggests that the second file contains the private key required for decryption.
9. The function decrypts the message (`msg`) using the loaded private key (`private`) and the **rsa.decrypt()** function.
10. The decrypted message is returned after decoding it from bytes to a string using **.decode()**.

7. Elliptic Curve Digital Signature Algorithm (ECDSA):

```
from pycoin.ecdsa import generator_secp256k1, sign
import hashlib

# msg = 'QmcJVKBZuSLuVu6sW7AwPbDMMWu4imR6oCxWFBverWdcXw'

# privKey = 345678909876543456789765434567896543456789

def signm(msg, privKey, generator_secp256k1, hashlib, sign):

    hashBytes = hashlib.sha3_256(msg.encode("utf8")).digest()
    hashBytes = int.from_bytes(hashBytes, byteorder="big")
    signature = sign(generator_secp256k1, privKey, hashBytes)
    return signature
```

Fig 7.7: Code snippet of the digital signature function

The code snippet you provided demonstrates how to sign a message using the elliptic curve digital signature algorithm (ECDSA) with the secp256k1 curve. Here's an explanation of the code:

1. The code imports necessary modules and functions: **generator_secp256k1** from **pycoin.ecdsa**, **sign** from **pycoin.ecdsa**, and **hashlib** for hashing operations.
2. The **signm** function is defined, which takes three parameters: **msg** (the message to be signed), **privKey** (the private key to sign the message with), and **generator_secp256k1** (the elliptic curve generator).
3. Inside the **signm** function, the message is first hashed using the SHA3-256 algorithm from the **hashlib** module. The resulting hash is stored in **hashBytes**.
4. The **hashBytes** are then converted to an integer using the **int.from_bytes()** method, specifying the byte order as "big".
5. The **sign** function from **pycoin.ecdsa** is called with the **generator_secp256k1**, **privKey**, and **hashBytes** as arguments. This function performs the elliptic curve digital signature operation and returns a signature.
6. The signature is then returned from the **signm** function.
7. To use this code, you need to provide a valid **msg** (message) and **privKey** (private key) to sign the message. The **generator_secp256k1** should be a valid elliptic curve generator for the secp256k1 curve. The code assumes that the **hashlib** module supports SHA3-256 hashing.

8. Verifying authenticity of file using digital signature:

```
def verify_file():
    fileHash = request.args.get('filehash')
    print(type(fileHash))
    print(fileHash)
    filedata = db.filedata.find({"filehash":fileHash})
    print(filedata)
    for f in filedata:
        signature = (f["signature"])
        to = f["to"]
    print(signature)
    signature = eval(signature)
    print(type(signature))
    pub = db1.userdata.find({"name" : to})
    for f in pub:
        pubKey = (f["pubKey"])
    pubKey = eval(pubKey)
    print(type(pubKey))
    print(pubKey)

    valid = verifym(fileHash,pubKey,generator_secp256k1,hashlib,signature,verify)
    # hashBytes = hashlib.sha3_256(fileHash.encode("utf8")).digest()
    # hashBytes = int.from_bytes(hashBytes, byteorder="big")
    # validated = verify(generator_secp256k1, pubKey, fileHash, signature)
    # print(validated)
    print(valid)
    if valid:
        message = "Valid Document Signed by\t"+to
        return render_template("Verify.html",message = message)
    else:
        message = "Invalid Document"
        return render_template("Verify.html",message = message)

    return render_template("Verify.html")
```

Fig 7.8: Code snippet of file verification

The **verify_file** function is responsible for verifying the authenticity of a file using a digital signature. Here's an explanation of the code:

1. The function retrieves the **fileHash** parameter from the request's query parameters. It represents the hash of the file to be verified.
2. The function queries the database (**db.filedata**) to find the relevant file data associated with the given **fileHash**.
3. It extracts the **signature** and **to** (recipient) information from the file data. The **signature** is stored as a string in the database and is evaluated using **eval()** to convert it back to its original type (e.g., a tuple or bytes object).
4. The function queries the database (**db1.userdata**) to find the public key (**pubKey**) associated with the recipient (**to**).
5. The **pubKey** is evaluated using **eval()** to convert it back to its original type.

6. The function calls the **verify** function, passing the **fileHash**, **pubKey**, **generator_secp256k1** (presumably an elliptic curve generator), **hashlib** module, **signature**, and **verify** function. This suggests that the verification process involves the use of an elliptic curve algorithm and a custom-defined **verify** function.
7. The result of the verification process is stored in the **valid** variable.
8. Depending on the value of **valid**, the function generates an appropriate message indicating whether the document is valid or invalid.
9. The generated message is passed to the "Verify.html" template for rendering.
10. Finally, the rendered template is returned as the response.

9. Face Recognition module:

```
import face_recognition
import numpy as np
import os
from mongo_files import get_image, delete_all
from pymongo import MongoClient

# # Load a sample picture and learn how to recognize it.
# krish_image = face_recognition.load_image_file("Krish/krish.jpg")
# krish_face_encoding = face_recognition.face_encodings(krish_image)[0]

# # Load a second sample picture and learn how to recognize it.
# bradley_image = face_recognition.load_image_file("Bradley/bradley.jpg")
# bradley_face_encoding = face_recognition.face_encodings(bradley_image)[0]

# Create
known_face_encodings=[]
known_face_names=[]

camera = cv2.VideoCapture(0)

for dirpath, dirname, filenames in os.walk(os.getcwd()+os.getcwd()+r'\\images_from_mongo_training\\'):
    for f in filenames:
        image = face_recognition.load_image_file(f)
        face_encoding = face_recognition.face_encodings(image)[0]
        known_face_encodings.append(face_encoding)
        known_face_names.append(f)

face_locations = []
face_encodings = []
face_names = []
process_this_frame = True
```

Fig 7.9: Code snippet of face recognition module

The Above code snippet demonstrates the usage of the `face_recognition` library for face recognition tasks. Here's an explanation of the code:

1. The necessary imports are made, including **face_recognition** for face recognition operations, **numpy** for array manipulation, **os** for file and directory operations, and other modules for MongoDB integration.
2. The code initializes an empty list **known_face_encodings** and **known_face_names** to store the face encodings and corresponding names.
3. The code sets up a video capture using OpenCV's **VideoCapture** function to access the webcam.
4. The code loops through the files in the specified directory (assumed to contain face images for training). For each file, it loads the image using **face_recognition.load_image_file** and computes the face encoding using **face_recognition.face_encodings**. The face encoding is then appended to **known_face_encodings**, and the file name is appended to **known_face_names**.

10. Video streaming and capturing frames:

```
class StreamingVideoCamera(object):
    def __init__(self, username):
        self.video = cv2.VideoCapture(0)
        self.username = username
        (self.grabbed, self.frame) = self.video.read()
        threading.Thread(target=self.update).start() #ran as a different thread to avoid getting stuck in the while true

    def __del__(self):
        self.video.release() #release all resources once object is destroyed

    def get_frame(self, id):
        image = self.frame #recieve the frame
        os.chdir(os.getcwd()+'\\images')
        cv2.imwrite(str(self.username)+str(id)+".jpg", image)
        os.chdir(os.getcwd()+'\\..')
        _, jpeg = cv2.imencode('.jpg', image) #converts (encodes) image formats into streaming data and stores it in-memory cache. It is mostly used to compress image data for
        return jpeg.tobytes()

    def update(self):
        while True:
            (self.grabbed, self.frame) = self.video.read() #func to read from the video camera so that it can be displayed

def gen(camera):
    i=0
    while i<10:
        time.sleep(1)
        i+=1
        frame = camera.get_frame(i)
        if i==10:
            add_files_to_mongo(camera.username)
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
```

Fig 7.10: Code snippet to capture the streaming video

The code you provided defines a **StreamingVideoCamera** class that represents a video camera for streaming and capturing frames. Here's an explanation of the code:

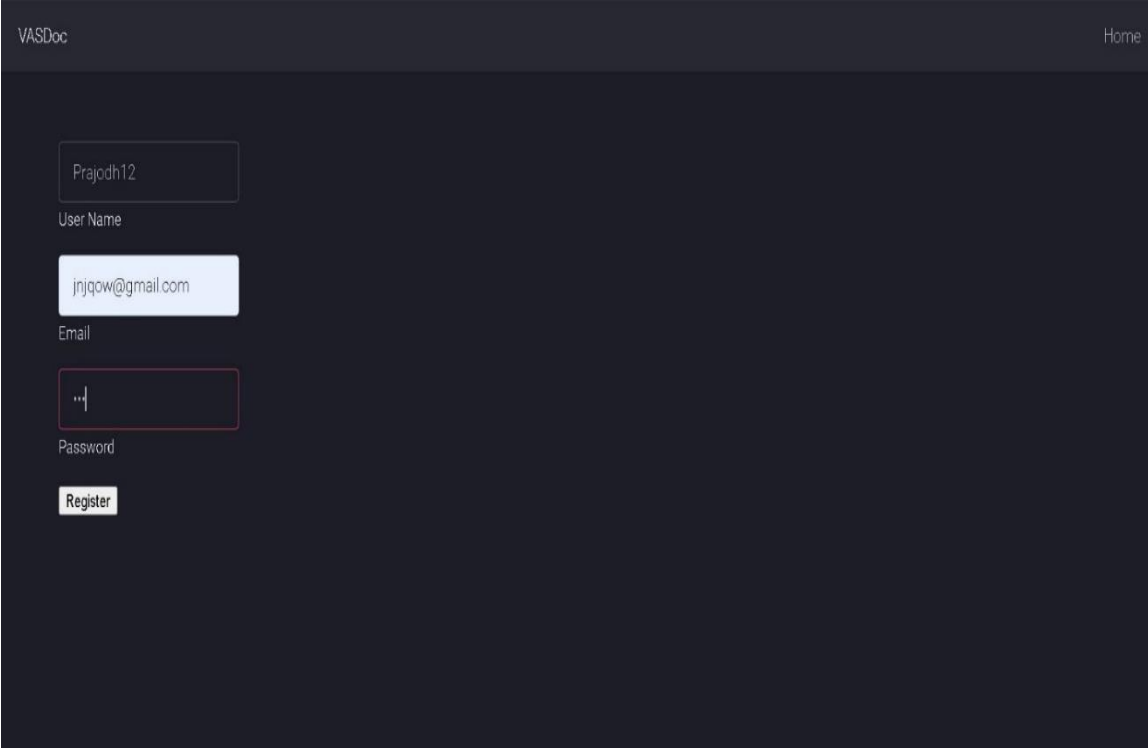
1. The **StreamingVideoCamera** class is initialized with the **username** parameter, which is used to identify the user associated with the camera.
2. In the constructor (**__init__** method), the class sets up the video capture using **cv2.VideoCapture(0)** to access the webcam. It also initializes the **grabbed** and **frame** variables to store the status and the current frame of the video capture.
3. The **__del__** method releases all resources associated with the video capture when the object is destroyed.
4. The **get_frame** method retrieves a frame from the video capture and saves it as an image file using the **cv2.imwrite** function. The image file is named based on the **username** and an **id** parameter provided to the method. The method then converts the frame to JPEG format and returns the JPEG data as bytes.
5. The **update** method runs in a continuous loop and continuously updates the **grabbed** and **frame** variables by reading frames from the video capture using **self.video.read()**.
6. The **gen** function takes a **camera** object as input and generates video frames in a streaming format. It uses a loop to capture frames from the camera and yield the frames as streaming data. After capturing a certain number of frames (in this case, 10 frames), it calls the **add_files_to_mongo** function and then stops capturing frames.

7.2 Result with Screenshots:



Fig 7.11 Website welcome page

A website's welcome page is the first page that visitors see when they visit your website. The goal of a welcome page is to create a positive first impression and give visitors an idea of what your website is about. Hence, we made it look attractive and imparted knowledge about what we were trying to do.



VASDoc Home

Prajodh12
User Name

jnjqow@gmail.com
Email

...
Password

Register

Fig 7.12: Registration Page

On this page, the user can enter three fields, his username, his valid email ID, and his password. We had checks to prevent users from providing an invalid email ID. We also don't allow users to register with a username already in use.

On clicking "register," the user is redirected to the image registration page of the website to capture biometrics

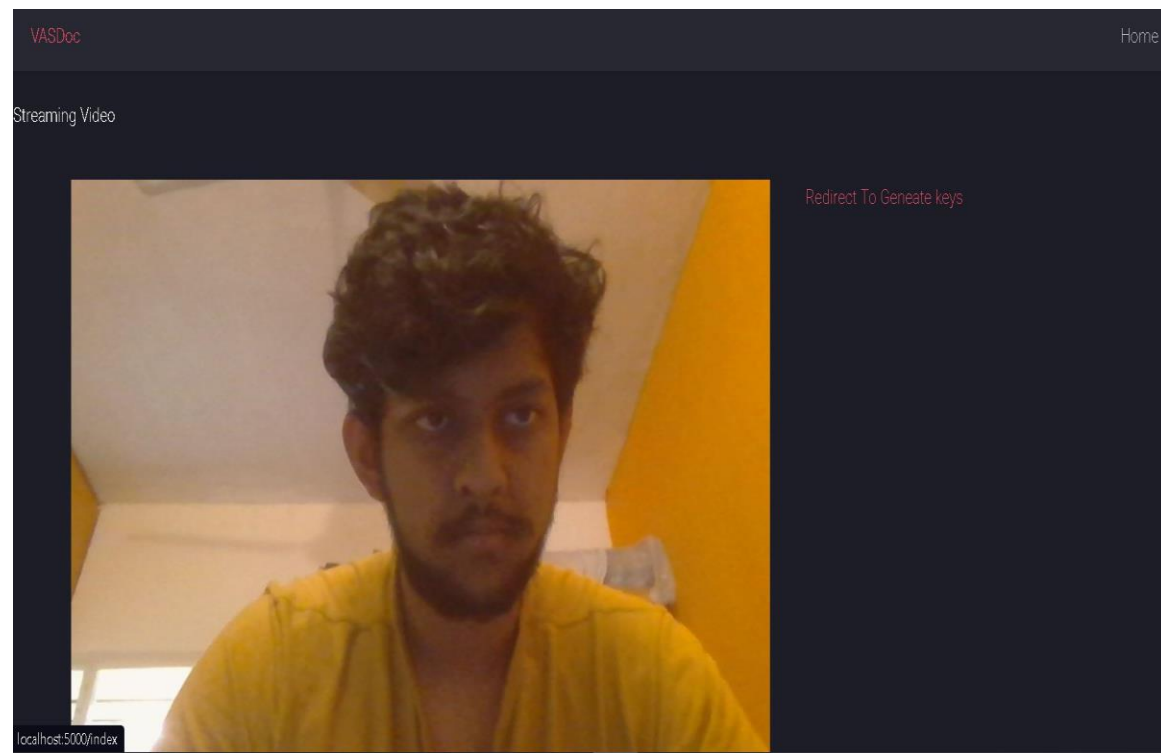


Fig 7.13 Image Registration

Image registration websites typically allow users to upload images and compare them to other images, with the goal of finding similarities or differences between the images. We are registering the user's images (face images) 10 images of the user are uploaded to. MongoDB is used in two ways to generate keys for digital signatures and second face recognition to log in to the website.

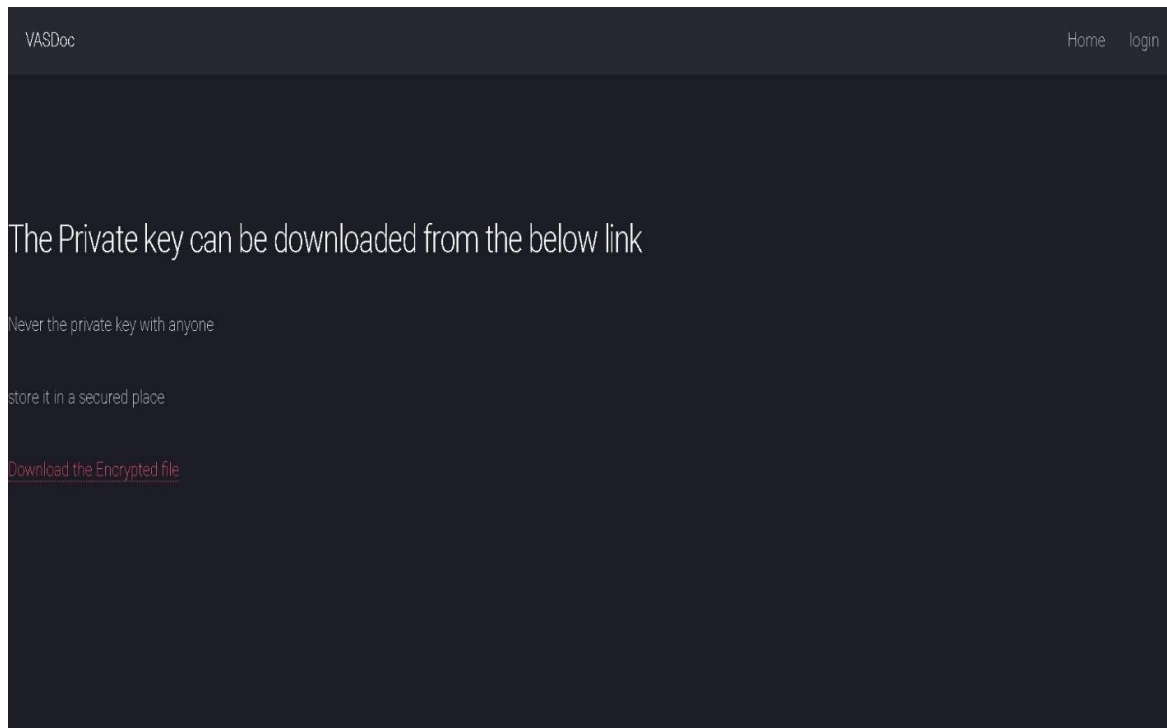


Fig 7.14 Generate your Private Key

A private key is a secret code that is used to unlock and access a particular cryptocurrency wallet or digital asset. It is a long string of numbers and letters that is mathematically related to a public key. Each user gets his own personal private key, which he can download and use for file uploads.

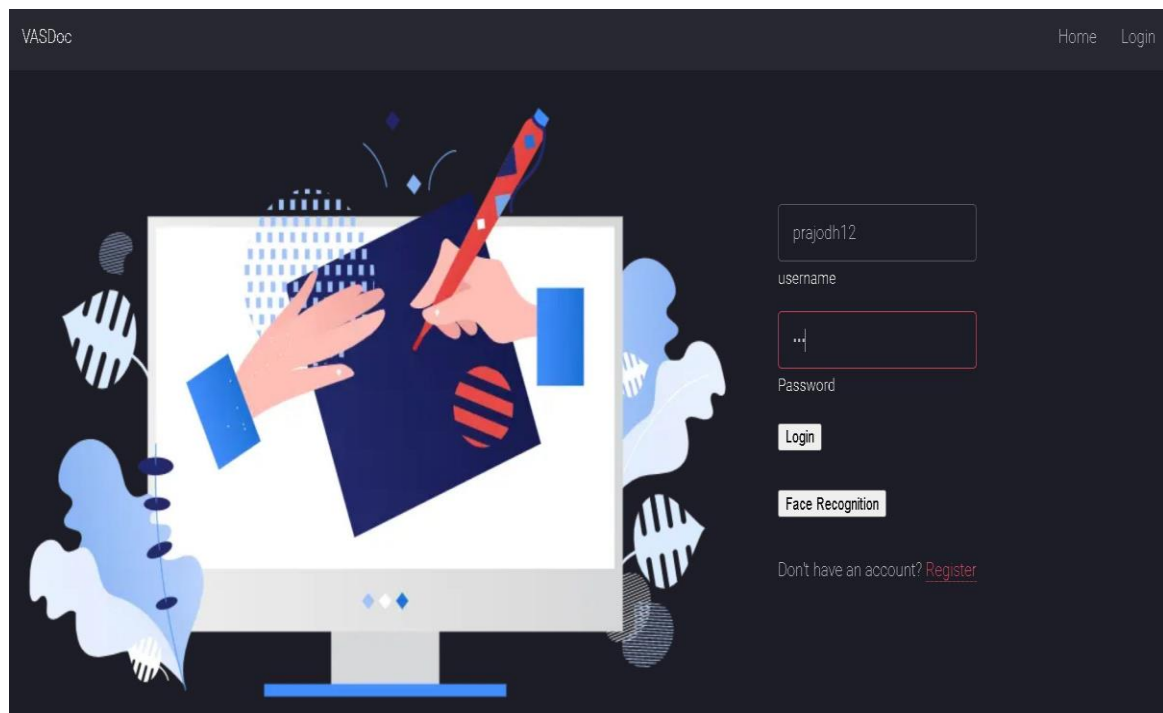


Fig 7.15 Login Page

A login page is a web page that allows users to authenticate themselves by providing a username and password or other credentials. Login pages are commonly used in web applications and online services to restrict access to authorized users only. A typical login page usually consists of a form with fields for the username and password, and sometimes additional security measures such as two-factor authentication. Once the user submits their credentials, the application or service verifies them.

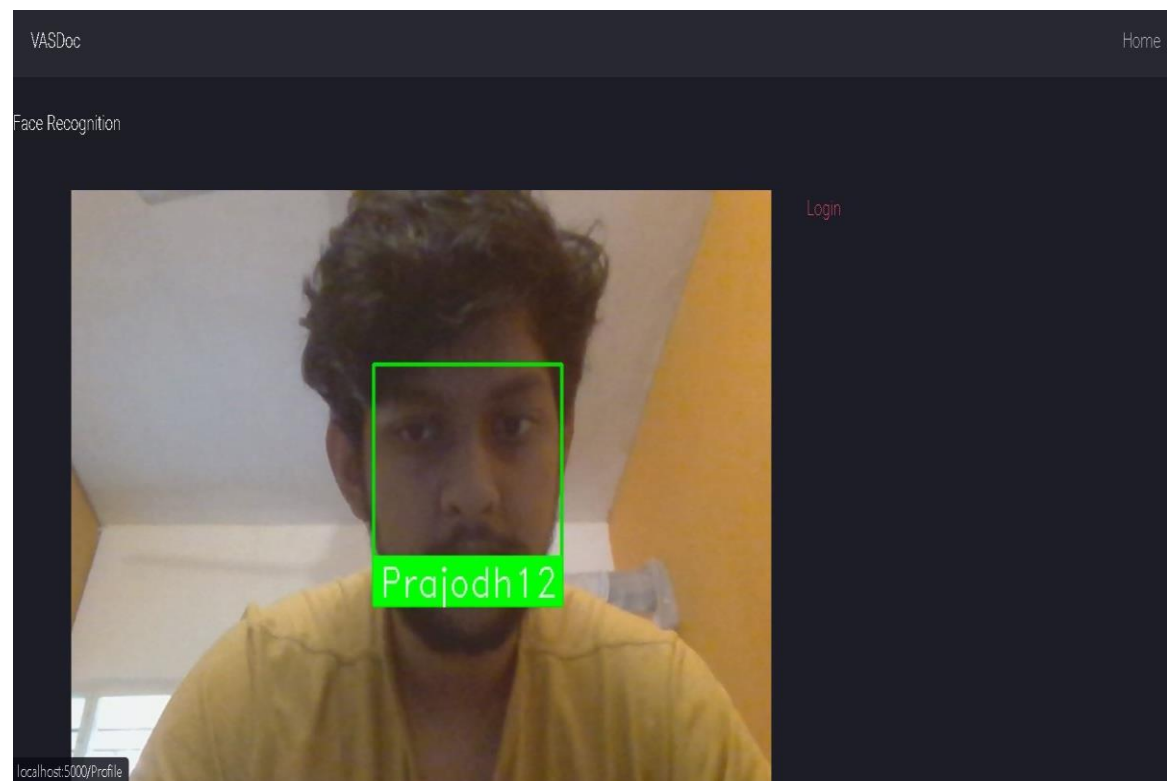


Fig 7.16 Face Recognition

We're using a built-in TensorFlow model for image classification, which has proven to have an accuracy of 98.89. It converts images into encodings and compiles these encodings from the input stream of images coming in to detect faces.

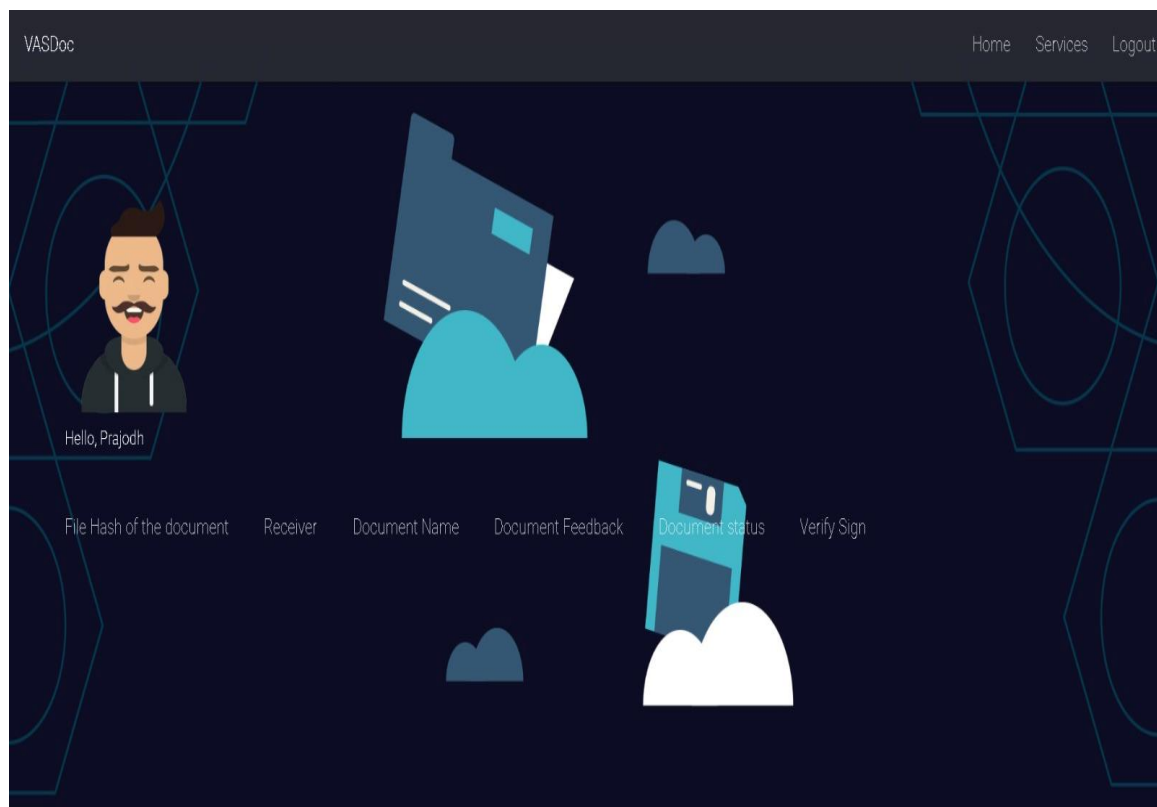


Fig 7.17 Profile Page

This page is to show the user and his activities, including all his uploads and their details. The details include the File hash (CID), Receiver name, Document name, Any feedback sent by the receiver concerning the document, Status of the document – if it is approved or not and finally an option to check if the document is digitally signed or not. Since this is a newly created user and doesn't have any uploads, it is blank.

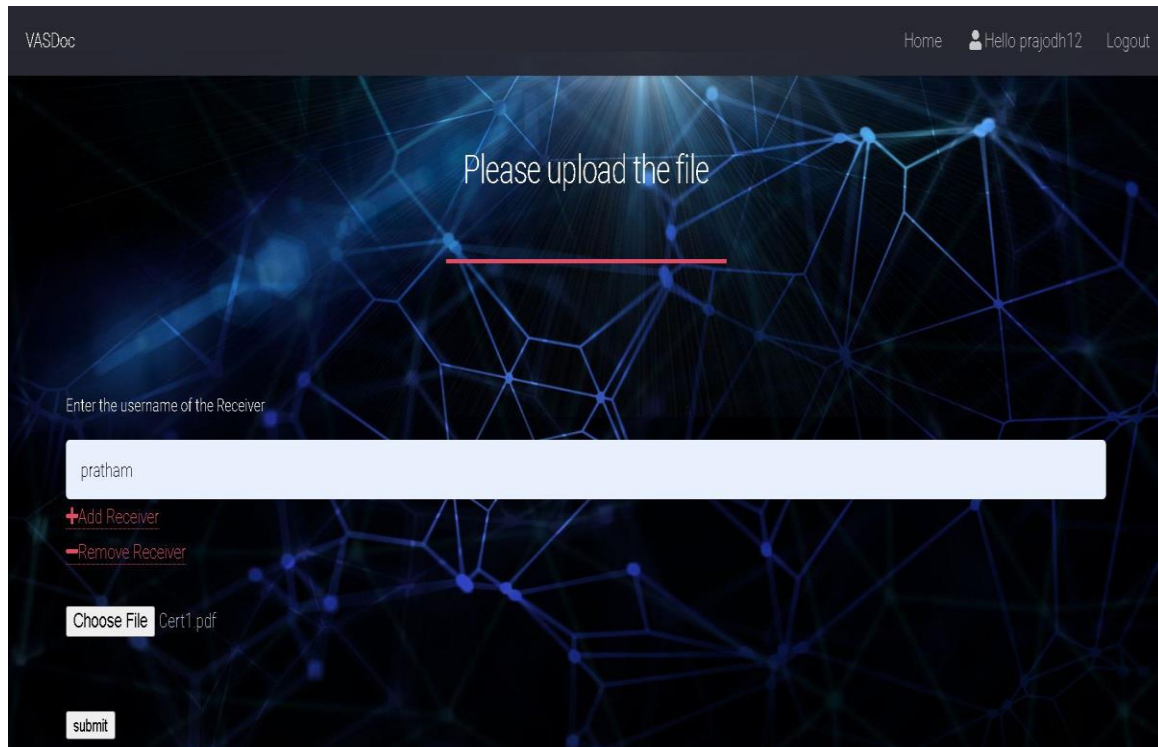
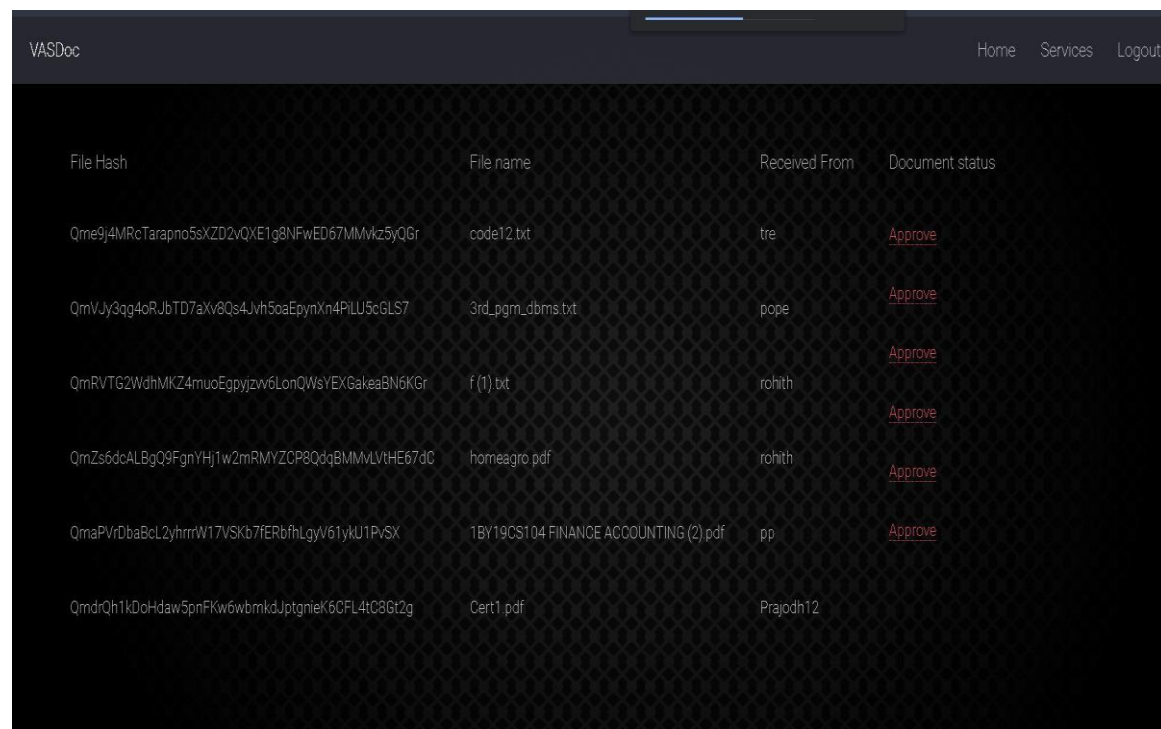


Fig 7.18 File Upload Page

Here in file upload page, we allow the users to upload file to any number of receivers, the file will be encrypted and uploaded to the ipfs server, so that the integrity of the file can be maintained. The ipfs server is protocol which works on the principle of block chain.

In this case, we are sending it to a receiver named Pratham who has already registered in our website.



File Hash	File name	Received From	Document status
Qme9J4MRcTarapno5sXZD2vQXE1g8NFwED67MMVvz5yQGr	code12.txt	tre	Approve
QmVJy3qg4oRJbTD7aXv8Qs4Jvh5oaEpyXn4PILU5cGLS7	3rd_pgm_dbms.txt	pope	Approve
QmRVTG2WdhMKZ4muoEgpyjzw6LonQWsvYEXGakeaBN6KGr	f (1).txt	rohith	Approve
QmZs6dcALBgQ9FgnYHj1w2mRMVZCP8QdqBMMVLvtHE67dC	homeagro.pdf	rohith	Approve
QmaPvrdBaBcL2yhrmW17VSKb7FERbfhLgyV61ykU1PvSX	1BY19CS104 FINANCE ACCOUNTING (2).pdf	pp	Approve
QmdrQh1kDoHdaw5pnFKw6wbmrkdJptgniek6CFL4tC8Gt2g	Cert1.pdf	Prajodh12	

Fig 7.19 Received Files

The files uploaded by the sender show up in this page and the receiver could view the file by downloading the file, If the file uploaded by the sender meets the expectations, then the receiver will sign and approve the document. If the file doesn't meet the receiver's expectation, then the receiver will disapprove the document. If the user has some suggestions, then he can even comment on the page.

In this case, the user has received multiple files from different users registered in our website.



Fig 7.20: Adding comments to the file

In this page of the website, if there are any changes that needs to be made to file or any discrepancies in the received file, the receiver can add comments (in the comment box provided) regarding the changes that needs to be incorporated and send back the file.



Fig 7.21: Approving the file using digital signature

Once the changes are made (If there were any) and the document is ready for finalization, The user can digitally sign the document by using his private key and click on the approve and sign option. He can still add any comments if needed.

On submission, the customer signs the document digitally, which is now sent back to the sender of the file.

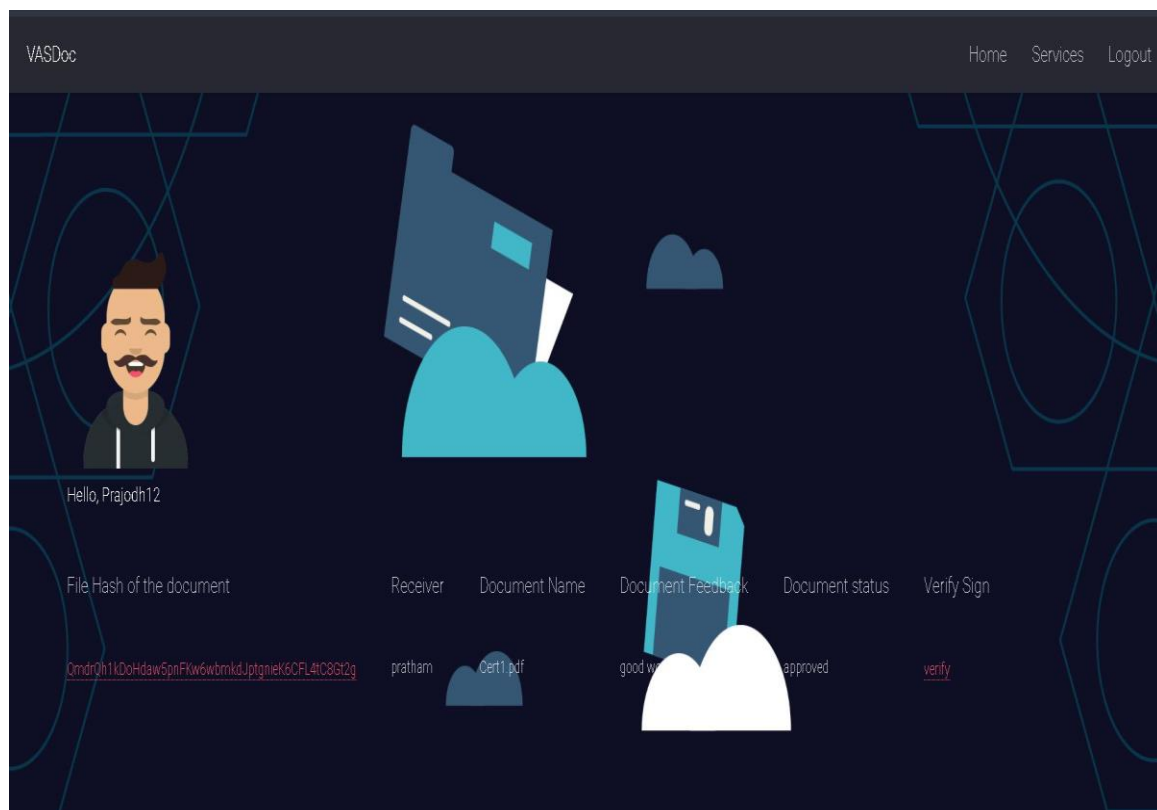


Fig 7.22: Checking the document status.

In this page of the website, the sender can check the status of the uploaded file. If the document is approved, it'll display the document status as 'Approved' else it'll display 'None'.

If approved, we can verify the authenticity of the digital signature on clicking on 'verify'.

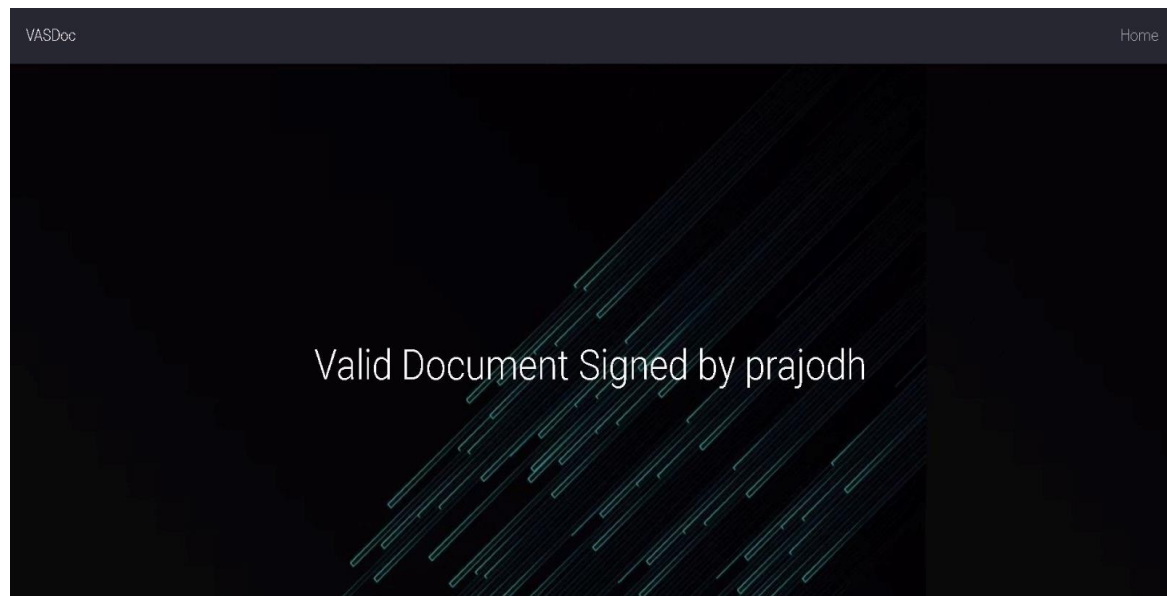


Fig 7.23: Valid Document

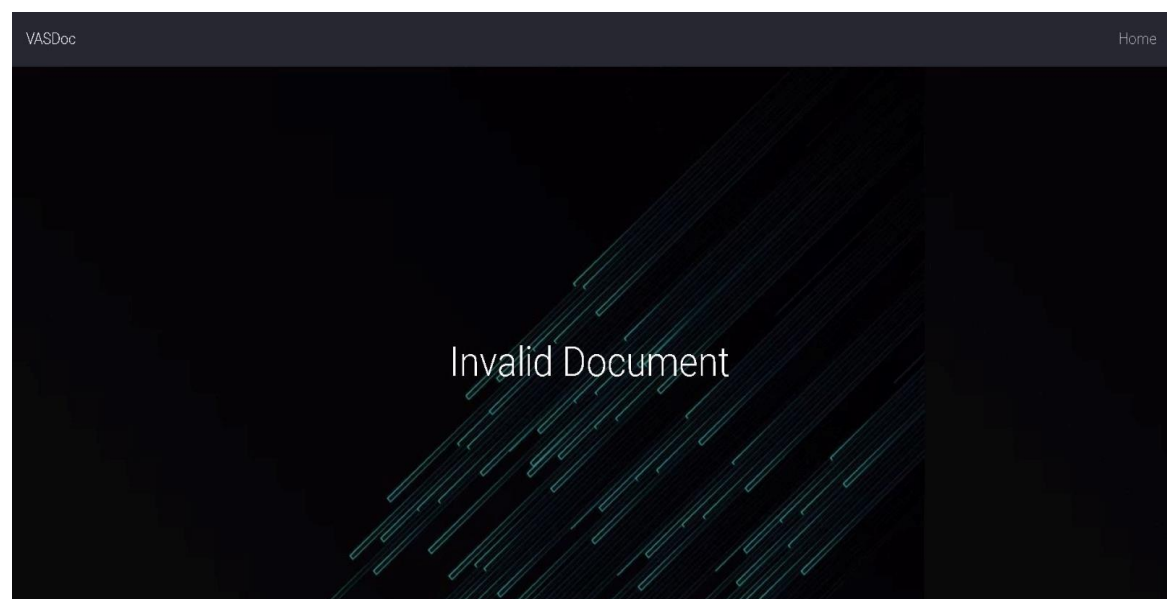


Fig 7.24: Invalid Document

Finally, When the user clicks on the 'verify' link, he is redirected to this page where, the user can check the authenticity of the person, i.e., who signed the document he sent to. If it's a valid signature, it's a valid document.

Here a text is displayed if it is a valid document (in fig 7.23) or else it shows invalid document (in fig 7.24).

CHAPTER 8

CONCLUSION

As presented, the existing systems for exchanging digital documents are centralized systems, thus these systems are vulnerable to be unavailable for users due to many problems such as (single-point of failure, natural disasters, privacy issues etc..). Using a decentralized system such as IPFS can solve the problems of a single point of failure and natural disasters.

This project has focused on the development of a system for biometric digital signing and exchange of documents using IPFS. The system utilizes facial recognition technology for biometric signing and IPFS for document exchange, ensuring secure and reliable transmission of sensitive documents. The use of elliptic curve cryptography ensures stronger security compared to traditional RSA encryption methods.

Throughout the course of the project, we have explored the limitations of existing systems and the motivation for using IPFS and biometric authentication. We have also discussed the software requirements and specifications, as well as the challenges faced during the development process.

Overall, the system has the potential to revolutionize the way documents are exchanged and signed, particularly in industries such as finance, healthcare, and legal services. It provides a secure and reliable platform for transmitting sensitive documents, while also enhancing the efficiency and speed of the signing process.

REFERENCES

- [1] “# What Is Ipfs.” IPFS Docs, n.d. <https://docs.ipfs.tech/concepts/what-is-ipfs/>.
- [2] Krishnan, Armin (2020). "Blockchain Empowers Social Resistance and Terrorism Through Decentralized Autonomous Organizations". *Journal of Strategic Security*. 13 (1): 41–58. doi:10.5038/1944-0472.13.1.1743.
- [3] *Interplanetary file system* (2023) *Wikipedia*. Wikimedia Foundation. Available at: https://en.wikipedia.org/wiki/InterPlanetary_File_System.
- [4] “Reading Your Face: How Does Facial Recognition Work?” ARATEK, n.d. <https://www.aratek.co/news/how-does-facial-recognition-work>.
- [5] Asraa Ahmed, Taha Hasan, Firas A. Abdullatif, Mustafa S. T, Mohd Shafry Mohd Rahim (2019). "A Digital Signature System Based on Real Time Face Recognition". *IEEE 9th International Conference on System Engineering and Technology (ICSET 2019)*, 7 October 2019, Shah Alam, Malaysia. 2 (2): 17-24.doi: 10.1109/ICSEngT.2019.8906410.
- [6] Bo Fu, et al. “A Dynamic Resource Management Scheme for Content-Based P2P Networks: A Case Study of IPFS”. *IEEE Access*, 9:75515-75526, 2021.
- [7] Yasmeen shaher Alsman and Anas Abu Taleb, “Exchanging Digital Documents Using Blockchain Technology”, Jordan, 2021.
- [8] Aastha Chowdhary, Shubham Agrawal and Dr. Bhawana Rudra, “Blockchain based Framework for Student Identity and Educational Certificate Verification”, India, 2021.
- [9] Nikita I. Chesnokov, Denis A. Korochentsev, Larissa V. Cherckesova, Olga A. Safaryan, Vladislav E. Chumakov, Irina A. Pilipenko, “Software Development of Electronic Digital Signature Generation at Institution Electronic Document Circulation”, Russia, 2020.
- [10] Anastacio Antolino Hernández, Juan Carlos Olivares Rojas, “Management of digital documents with encrypted signature, through the use of centralized PKI, and distributed using blockchain for a secure exchange”, Mexico, 2019.
- [11] R. P. Pasupulati and J. Shropshire, "Analysis of centralized and decentralized cloud architectures," *SoutheastCon 2016*, Norfolk, VA, USA, 2016, pp. 1-7, doi: 10.1109/SECON.2016.7506680.
- [12] IvanOnTech. “Interplanetary File System Explained - What Is Ipfs?” *Moralis Academy*, December 16, 2021. <https://academy.moralis.io/blog/interplanetary-file-system-explained-what-is-ipfs>.

- [13] Doan, Trinh Viet, Vaibhav Bajpai, Yiannis Psaras, and Jörg Ott. "Towards decentralised cloud storage with IPFS: Opportunities, challenges, and future directions." *arXiv preprint arXiv:2202.06315* (2022).
- [14] Benet, Juan. "Ipfs-content addressed, versioned, p2p file system." *arXiv preprint arXiv:1407.3561* (2014).
- [15] IvanOnTech. "Interplanetary File System Explained - What Is Ipfs?" Moralis Academy, December 16, 2021. <https://academy.moralis.io/blog/interplanetary-file-system-explained-what-is-ipfs>.
- [16] Faundez-Zanuy, Marcos. "Biometric security technology." *IEEE Aerospace and Electronic Systems Magazine* 21, no. 6 (2006): 15-26.
- [17] Silicon Mechanics five advantages of IPFS and how to achieve them. Available at: <https://www.siliconmechanics.com/news/5-advantages-of-interplanetary-file-system>.
- [18] Garcia, A. (2023) What is biometric authentication? Klippa. Available at: <https://www.klippa.com/en/blog/information/biometric-authentication/>.
- [19] Rastogi, Neha. "Biometrics Technology and Its Scope in Future." Engineers Garage, n.d. <https://www.engineersgarage.com/biometrics-technology-and-its-scope-in-future/>.
- [20] "Patent, #8220;System, and, Method, Digitally, Signing, Documents, Using, Biometric, Data, in, a, Blockchain, or, PKI, #8221;," PatentPC. @PatentPC, January 6, 2023. <https://www.patentpc.com/blog/patent-for-system-and-method-for-digitally-signing-documents-using-biometric-data-in-a-blockchain-or-pki-us10516538b2>.
- [21] Editor. "Non-Functional Requirements: Examples, Types, How to Approach." AltexSoft. AltexSoft, February 12, 2020. <https://www.altexsoft.com/blog/non-functional-requirements/>.
- [22] "Why Python Is the Best Programming Languages for Web Development." Probytes Web Development Company, June 11, 2018. <https://www.probytes.net/blog/python-web-development/>.
- [23] "Introduction to Flask." Introduction to Flask - Python for you and me 0.5.beta1 documentation, n.d. <https://pymbook.readthedocs.io/en/latest/flask.html>.
- [24] Cobb, Michael. "What Is the RSA Algorithm? Definition from Searchsecurity." Security. TechTarget, November 4, 2021. <https://www.techtarget.com/searchsecurity/definition/RSA>.

- [25] Rebecca. "RSA Encryption Explained – Everything You Need to Know." History, March 3, 2023. <https://history-computer.com/rsa-encryption/>.
- [26] Frankel, Sheila, Rob Glenn, and Scott Kelly. *The AES-CBC cipher algorithm and its use with IPsec*. No. rfc3602. 2003.
- [27] Ch Sekhar, P Sai Meghana (2020). "A Study on Backpropagation in Artificial Neural Networks". Asia-Pacific Journal of Neural Networks and Its Applications Vol.4, No.1 (2020), pp.21-28. doi: 10.21742/AJNNIA.2020.4.1.03
- [28] Pradeep. "Building Your First Neural Network with Tensorflow – Deep Learning 2." The Geek's Diary. The Geeks Diary, March 24, 2023. <https://thegeeksdiary.com/2023/03/24/building-your-first-neural-network-with-tensorflow-deep-learning-2/>.
- [29] "Blockchain - Elliptic Curve Cryptography." GeeksforGeeks. GeeksforGeeks, November 17, 2022. <https://www.geeksforgeeks.org/blockchain-elliptic-curve-cryptography/>.
- [30] Nafisah, Zumrotun & Rachmadi, Muhammad & Imah, Elly. (2018). Face Recognition Using Complex Valued Backpropagation. Jurnal Ilmu Komputer dan Informasi. 11. 103. 10.21609/jiki.v11i2.617. Callan, T. and France, N. (no date) *RSA, DSA and ECC encryption differences / Sectigo® Official*. Available at: <https://sectigo.com/resource-library/rsa-vs-dsa-vs-ecc-encryption>.