National College of Ireland

Enhanced File Transfer Security in Django Web Applications with TOTP-Based Multi-Factor Authentication and Blowfish/AES Encryption on AWS Cloud

MSc Research Project
Master of Science in Cloud Computing

NIKHIL DEVABHAKTUNI
Student ID: x22156411

School of Computing
National College of Ireland

Supervisor: Shreyas Setlur Arun

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	NIKHIL DEVABHAKTUNI
Student ID:	x22156411
Programme:	Master of Science in Cloud Computing
Year:	2024
Module:	MSc Research Project
Supervisor:	Shreyas Setlur Arun
Submission Due Date:	25/04/2024
Project Title:	Enhanced File Transfer Security in Django Web Applications with TOTP-Based Multi-Factor Authentication and Blowfish/AES Encryption on AWS Cloud
Word Count:	6718
Page Count:	22

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Nikhil Devabhaktuni
Date:	27th May 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Enhanced File Transfer Security in Django Web Applications with TOTP-Based Multi-Factor Authentication and Blowfish/AES Encryption on AWS Cloud

NIKHIL DEVABHAKTUNI
x22156411

Abstract

Securing web applications and sensitive data stored on the cloud is critical to prevent breaches. Django web frameworks lack native security capabilities making apps vulnerable which necessitates the need for multi-layered authentication systems to harden security. This research implements a multi-layered security solution to improve data protection in a Django application by combining Time-based One Time Password (TOTP) with the default Django authentication process and the use of blowfish and AES encryption algorithms for securing the file transfer to Amazon Web Services (AWS) S3 storage bucket. The inclusion of TOTP adds an additional verification layer after logging with user credentials, requiring users to enter a one-time code that expires in time from an authenticator app. Blowfish, chosen for its variable key length and AES, chosen for its larger block size are used to encrypt the files providing strong security. The encrypted files are then transferred to S3 buckets with strict access control permissions to prevent unauthorized access. The results show that this defense approach substantially boosted the Django application data security by preventing a hacker from getting access to the account with compromised credentials as TOTP provides additional authentication layer. The performance of Blowfish and AES are also evaluated in terms of execution times and compression ratios to identify the best candidate for cloud data transfers. This system with multi-factor authentication, strong encryption, and secure AWS cloud storage works seamlessly to prevent unauthorized access and protect against various threats to Django applications and data hosted on the cloud.

1 Introduction

Web applications built using frameworks like Django enable rapid development but lacks native security capabilities making web applications built using the framework susceptible to various cyber threats and attacks like cross-site scripting, SQL injections, remote code execution, etc (Gupta et al.; 2022). With increasing cybercrimes, it becomes imperative to harden the security of Django apps to protect sensitive user data as well as ensure availability and integrity of services. Additionally, with rapid cloud adoption, web applications now integrate with cloud platforms like AWS for hosting backend infrastructure and storing critical data assets. The distributed architecture of cloud introduces new

attack surfaces making data hosted on cloud storage vulnerable to breaches due to issues like data leaks, hijacking of accounts, insider threats, etc. Thus, enhancing the security of Django web applications along with securing data storage and transfers on cloud is crucial (Veeresh and Parvathy; 2022). Multi-layered solutions need to be implemented spanning across user access controls, cryptography, cloud security configurations following the defense-in-depth approach to harden the web apps and cloud data against cyber threats.

1.1 Problem Statement

The default Django framework lacks native authentication, access controls and cryptography capabilities. Passwords stored in cleartext are prone to offline dictionary attacks along with the risk of credentials stuffing to gain unauthorized access. Further, cloud storage data can be compromised due to issues like data leaks, hijacking of accounts by insiders in cloud provider environments. According to Acunetix *Acunetix* (n.d.) research, around 39% of web applications contain high severity vulnerabilities and the average time to remediate them is 197 days after discovery. Verizon's 2021 Data Breach Investigations Report *Verizon* (n.d.) also found that web application attacks continue to dominate with over 25,000 incidents. High-profile examples include the DoorDash data breach *Mashable* (n.d.) affecting 4.9 million users and the Printful customer data breach impacting thousands of accounts. Lack of multi-factor authentication and unencrypted data were key issues.

This necessitates implementing multi-factor authentication in Django to strengthen user access controls by requiring an additional one-time password over and above the username/password. Cryptography algorithms need to be used to encrypt sensitive files before secure transfers to cloud storage for preventing data breaches. Hardening cloud security configurations through measures like identity access management, strict access controls on cloud storage buckets is also vital. There is a need for a multi-layered security solution spanning authentication, cryptography and cloud security to protect Django web apps and data hosted on cloud against adversaries. The approach should follow security best practices and defense-in-depth strategy to significantly enhance security and privacy.

1.2 Motivation

This research is motivated by the need to improve the security posture of web applications such that sensitive user data is protected against compromise and unauthorized access. While the Django framework itself provides user authentication, this relies solely on username and password making it vulnerable to credential stuffing or brute force attacks to gain entry. Adding a second-factor authentication using one-time passwords (OTPs) generated by apps like Google Authenticator requires the user to prove their identity using something they possess i.e. their smartphone. This mechanism is already widely adopted by financial, e-commerce, and technology platforms. Integrating Time-based OTP (TOTP) based multi-factor authentication improves the security of Django admin login and user login before access is granted. Furthermore, the Blowfish cipher provides strong symmetric key encryption suitable for encrypting data before network transfer and file storage. Using Blowfish encryption in Cipher Block Chaining (CBC) mode increases the complexity for attackers and limits data exposure. This research provides a blueprint and reference architecture to add these data security layers.

1.3 Research Question

To what extent can the overall security posture of Django web applications be strengthened by TOTP-based multi-factor authentication and Blowfish/AES encryption algorithms, with particular reference to user authentication and secure file transfers for data stored in AWS S3 buckets?

1.4 Research Objective

The objective of this research is to develop a multi-layered security solution for Django web applications and data hosted on AWS cloud that provides enhanced protection against cyber threats by combining Time-based One-Time Password TOTP multi-factor authentication and Blowfish symmetric encryption algorithm to provide confidentiality of sensitive files before secure transfer to Amazon S3 cloud storage.

1.5 Research Contributions

The salient contributions of this research include:

1. Integrate TOTP multi-factor authentication with the Django authentication system to strengthen access control by adding an extra layer of user identity verification.

2. Utilize Blowfish symmetric encryption algorithm to provide confidentiality of sensitive files before secure transfer to Amazon S3 cloud storage. Blowfish leverages variable length keys making it resistant to brute force attacks.

3. Strengthen AWS S3 cloud storage security configurations including access controls, logging, and user permissions to prevent unauthorized data access.

4. Implement a strategy across authentication, cryptography, and cloud security layers following security best practices to significantly improve Django application and data security.

1.6 Scope and Limitations

The scope of this research is limited to enhancing data security within Django framework-based web applications deployed on Amazon Web Services. While other authentication schemes like biometrics and hardware tokens are emerging options, only software-based TOTP has been chosen for implementation feasibility. The Blowfish cipher with CBC mode has been selected for its strong security properties, but the research is limited to symmetric encryption only and does not cover asymmetric public key encryption. The reference architecture demonstrates integration with Django and AWS services but does not extend to deployment mechanisms like Docker containers.

1.7 Thesis Structure

Section 1 presents the research projects, motivation, research question and objectives of the research. Section 2 details all the existing literature and their research gap that this research tries to cover. Section 3 provides the methodology for Django web application development using 2FA and Blowfish/AES encryptions. Section 4 details the design

specifications of the research problem covering system architecture, algorithms, s3 configuration and sequence flow diagrams. Section 5 provides the implementation of Django web application from user registration , 2FA, encryption and decryption process and s3 data upload and download. Section 6 showcases the hands-on deployment of the django web application in the local. Section 7 concludes the research with future scope.

2 Related Work

The impressive growth of cloud data storage systems with large enterprises and small businesses migrating their local data storage to the cloud has increased the possibility of unwarranted risks like data leaks, and privacy. The novelty of the research lies in proposing solutions to ensure privacy protection for users while utilizing cloud storage services.

2.1 Data Security in Cloud Services

A survey was presented by (Yang et al.; 2020) on secure data sharing and securing against leakages to highlight the limitations and challenges in data security on cloud storage systems. The primary focus was to avail privacy protection to the consumers of public and private cloud services through encryption mechanisms like attribute-based encryption (ABE), and identity-based encryption (IBE). Additionally, searchable encryption methods such as multi-keyword ranked search (MKRS) and searchable symmetric encryption (SSE) were also studied to enhance the user understanding, and adeptness and maintain data confidentiality in working with encrypted cloud data. (Sun; 2020) presented an extensive review on data security and privacy protection techniques where a framework was proposed with access control, ABE, and symmetric and asymmetric searchable encryption to evaluate different access control mechanisms, integrating the access controls and searchability for improved privacy. Future directions to access control were also discussed including dynamic fine-grained systems, trust-based access control, and space-time awareness models. The research was significant in the way that it led to a better comprehension and importance of privacy protection in cloud computing by stating the key technologies and future scope.

An exhaustive survey on the challenges associated with big data security and privacy due to the potential security threats with the storage of sensitive information on the cloud storage systems was presented in (Alouffi et al.; 2021). It discussed on the existing data privacy and security models, threats, and the challenges arising with it while analyzing recent models developed to mitigate these challenges and improve cloud data security. A systematic review was presented in (Riaz et al.; 2020) focusing on the security risks encountered by cloud service providers by outlining the threats and potential solutions to overcome those with the use of blockchains. The chief learnings from the study are: that data tampering emerged as one of the top most security issues, consumer grievances concerning service agreements and data policies, and the blockchain-based solution to address the shortcomings of the security mishaps in the cloud environments.

2.2 Django Web Application Framework

Notwithstanding the availability of a multitude of web application frameworks like Flask, Ruby on Rails, Express.js, and others, this research focuses on web application devel-

opment using Django, a Python-based opensource framework due to its high scalability, extensive support for third-party packages and built-in security tools that offers protection against common web attacks like SQL injection, cross-site request forgery (CSRF), etc., The authors in (Veeresh and Parvathy; 2022) focused on providing cybersecurity for data encryption for secure data transfers over the cloud and availing data access control through the use of private keys serviced through a Django web application. This study resolves to improve data security in cloud environments with the use of encryption and a web framework capable of providing the basic security features in-built. A real-time implementation of the web application was proposed in (Stigler and Burdack; 2020) with a focus on scalability, multiprocessing, and load optimization for faster response and evaluating its performance of the web application based on Django by applying queuing theory for scaling down the application based on arrival times, thereby contributing to stable server usage. This paper placed the importance on dynamic and efficient data handling and processing of information in real-time applications.

Duisebekova et al. (2021) presented in their research the security tools available in Django for data security in web services by demonstrating the services like CSRF-tokens, and cross-site scripting (XSS) protection inherent in Django. They also presented case studies that compared various attacks, and the protection scheme in Django to mitigate those attacks. This work showcased Django's abilities in protecting web applications through production configurations, HTTPS/SSL, and XSS attacks. A Django-based framework was presented in (Yu et al.; 2023) for improving house information management systems with user-centric approaches and real-time listing. The web application was developed with a focus on optimizing the user registration and login processes, restricting user roles in data handling on Redis and MySQL databases, and improving system configuration by utilizing Django's security offerings and Python's improved efficiency. The contribution of this research lies in the provisioning of a scalable, agile, lightweight, and effective housing data management based on Django.

A Django-based chat application for secure communication was presented in (Singh et al.; n.d.) by integrating various web tools like HTML, CSS, JavaScript, MySQL, and Django. The encryption was done using N-TEA (New Text Encryption Algorithm) for additional data security in securely transmitting chat messages and evaluating the efficiency of the algorithm in terms of encryption/decryption times and compression ratios. A comparative study was conducted between Django and Flask in (Ghimire; 2020) in developing a social network application using Flask and an e-commerce app with Django. The findings suggest, that though Flask offered simplicity in implementation and fine-grained control in configurations, Django had extensive library support to enhance the data security and scalability in large-scale deployments. The study concluded by presenting that both Flask and Django have their own merits and demerits in a real-time web application development use case, highlighting that only Django has the built-in security mechanisms to handle data security breaches and leaks in the context of cloud storage security.

2.3 Encryption Mechanisms for Cloud Security and Blowfish Algorithm

Singhal et al. (2022) proposed a cryptographic steganography technique for cloud data security using Blowfish and least significant bit (LSB) encryption to secure cloud data from unauthorized access, and preserve user privacy and integrity. The blowfish algorithm

was used to encrypt the message to be uploaded with a private key and LSB embedding was used to hide the message within a cover image. The research also compared the results of the proposed approach with other encryption algorithms like data encryption standard (DES), discrete cosine transforms (DCT), etc., in terms of key size, block size, and encryption times and found out that blowfish with LSB embedding performed well offering more flexibility making it an ideal candidate for data security in the cloud. A related comparison study of encryption algorithms like DES, 3DES, advanced encryption standard (AES), and Blowfish was presented in detail by (Radhi and Ogla; 2023) with cost and performance measures of encryption/decryption times, memory usage, optimal encoding bits, and entropy score as the evaluation criteria. This experimental framework that implements the algorithms was tested on a Fog server running an Ubuntu 16.04 distribution. It was found that Blowfish was thrice faster than others in comparison with least memory usage and best entropy value, an indicator of protection against cyberattacks, making it the right offering for web applications that are constrained by memory and runtimes.

The authors in (Singhal et al.; 2023) proposed an improved blowfish approach for plaintext and file encryption and compared it to DES, DCT, and AES encryption algorithms. This improved blowfish used a transformation model to reduce the rounds from the usual 16-round Feistel model and increased the block length to enhance security. This version of Blowfish was evaluated in terms of encryption/decryption times as the performance metric and outperformed AES and other algorithms, in addition to offering better security for cloud-shared messages and files. The research in (Seth et al.; 2021) presented a hybrid approach with Blowfish (symmetric encryption) and Paillier (homomorphic encryption) algorithms. The main objective was to decrease the total computation time and encrypted text size to save power and storage, respectively. The performance assessment was carried out utilizing factors such as quality of service (QoS) for different block cipher modes, and its calculation overhead. The results of this hybrid approach showed that Blowfish-Paillier-based encryption excelled in better security, storage, and computing speeds in comparison with RSA and AES justifying it as the perfect choice for cloud data security.

2.4 Multi-factor Authentication Techniques (MFA)

Reddy and Reddy (2018) presented an extensive review of multifactor authentication (MFA) approaches and their advantages compared to the use of static passwords in offering improved network security against password hacks and data leaks. This work was specifically focused on image-based MFA, fingerprint authentication, one-time passwords (OTPs), and time-based OTPs (TOTPs). It also explored how MFA can offer an additional layer of security for cloud data transfer and financial transactions by making hacking difficult to the least possible by necessitating multiple different ways of authenticating the credentials to an account. TOTPs are more secure in a way that they can avoid eavesdropping attacks by making the OTPs last only less than a minute in most user scenarios. The study concluded by suggesting the use of TOTPs offered by Microsoft/Google to generate unique passcodes for that added security boost for transactions or data transfer on the cloud.

The research presented in (Otta et al.; 2023) surveyed cloud-specific MFA techniques with new proposals like biometric authentication without any sophisticated hardware/software to overcome impersonation attacks. It also offered solutions to the existing

issues in the cloud services authentication and access control that covers both heterogeneous private clouds and edge clouds. It concluded that cloud data security can be significantly improved employing MFA-based user authentication with the inclusion of advanced facial recognition systems and recommended investigations into the cost and feasibility of heterogeneous cloud data centers. In the research presented by (Reese et al.; 2019), five two-factor authentication (2FA) methods were studied by following a practical subject-based laboratory experiment involving the usability and setup of 2FA implemented using SMS, TOTP, push notifications, etc., on a model banking website designed for this purpose. The daily login process was used as the observation point for the analysis of user preferences and 2FA usability which is the cornerstone of the success of MFA methods. The study found that the users preferred the use of hardware tokens and OTPs due to their ease of setup and usability over other approaches. The research concluded that with increased 2FA adoption, cloud-based security issues and privacy concerns can be warded off to instill the confidence of consumers to migrate to cloud platforms for data storage.

This section extends the discussion on MFA approaches and their integration into the Blowfish algorithm proposed in this research work. Durga et al. (2023) presented an MFA approach in conjunction with an optimized blowfish algorithm (OBA) for secure cloud data retrieval that improves data security hindering the adoption to cloud-based storage systems. The integration of MFA and OBA prevented unauthorized access to hackers and spoofers by guaranteeing only authorized users' access to data and with the inclusion of binary crow search for selecting the private keys for encryption, it also enhanced the protection against brute-force attacks. This combined strategy was found to be effective and a significant improvement over other existing approaches for secure cloud data security. The authors presented in (Umarani and Kumar; n.d.) a double encryption-based Blowfish (DEBF) for MFA with a stealth authentication mechanism to improve cloud data security. DEBF with MFA protects data by enforcing access control to prevent illegal access with better results compared to other encryption and authentication processes asserting their place in securing cloud computing data resources.

3 Methodology

This research proposes to develop a Django-based web application using two-factor (2FA) authentication done with Time-based OTP (TOTP) for the user authentication process to facilitate secure file transfers to the AWS S3 cloud storage. The methodology for the proposed model with its key components is presented in Figure 1.

TOTP based 2FA Mechanism: The Django authentication system integrates TOTP for improved user access controls by mandating an additional one-time password verification in tandem with the login and password credentials. This enables an additional level of authentication to gain access to the application. This is accomplished with the generation of a secret key linked to the user's account during the user registration process. This secret key can be used by authenticator apps like Google Authenticator, Microsoft Authenticator, Authy etc., to generate time-based one-time passwords (TOTP). During the login process, the Django authentication system attempts to validate the provided username and password credentials from the protected local database like MySQL. Once the username and password details are approved, the user is then directed to enter the TOTP code generated by the authenticator application that they have installed on their

mobile devices. The mobile devices themselves have in-built authentication methods like face unlock, fingerprint or passcode, thereby offering an additional layer of security in some unauthorized person gaining access to the TOTP application. The entered TOTP code is validated against the user's secret key using the TOTP library. If the provided TOTP code matches the expected value, it is considered as valid and the user is provided access to the application. An acknowledgement mail is also sent to the user's email address provided at the time of registration.

Encryption Mechanisms - Blowfish and AES: This primary role of the designed application is to provide the function of a cloud drive to which files can be uploaded and downloaded securely using encryption. While transferring files across the cloud network, ensuring the safety of the files where they are prone to interception or during the handling of the files where there is a possibility for unauthorized access is foremost essential. The use of cryptography to encrypt the files before transmission is therefore mandatory, especially when working with files that are shared across different places in a network. The most popular and powerful encryption algorithms are Blowfish and AES, both symmetric-key encryption algorithms, has been proposed to be used in this work to encrypt the files before the transfer. Blowfish operates on blocks of 64 bits and uses a key of variable length ranging from 32 bits to 448 bits. The AES algorithm works on blocks of 128 bits and allows for key sizes of either 128, 192, or 256 bits. Encryption keys are generated and securely stored in the database provider (MySQL) for the application. The choice between Blowfish and AES primarily depends on the application scope, like banking, e-commerce, content delivery etc., but finer control over these algorithms can be achieved by experimenting with different key lengths, flexible demonstrations of design, speed requirements, and others. The Django web application framework is utilized to perform encryption and decryption of the data using third-party python libraries that integrate well with the Django environment. Two modules were designed for the purpose of handling uploads (UploadFileAction) and downloads (DownloadFileAction) which provides the utilities to perform encryption and decryption using blowfish and AES, respectively as shown in Figure 1. The encrypted file details such as username, filename, and encryption type are saved to the application's local database (Auth DB) for further retrieval while decrypting the files. This manner of saving the file details along with the username and encryption type provides the option to restrict access to the files only to the authorised user.

AWS S3 Cloud Storage: The files are encrypted before being uploaded to the AWS S3 cloud storage, providing data protection during transfers and when stored. The choice of use of AWS for file storage is chiefly due to facts such as, cloud data center security, decoupled storage where the encrypted data is separated from the user credentials, enhancing security by preventing direct access to files from the database, encryption key management and compliance with industry standards. AWS also implements strict access controls for the access of S3 storage buckets using Identity and Access Management (IAM) policies or bucket policies. Only authorised users can upload or download files from the S3 bucket i.e., the user will be provided an option to download only those files that were uploaded by a specific user during a previous user login session. This ensures that there is no unauthorised access to unknown files or files uploaded by other users. This model of using local database like AuthDB to capture the user information, filename and encryption type before uploading the files can facilitate the process of downloading the file intended for a specific user with the proper decryption algorithm to retrieve the original contents of the file.

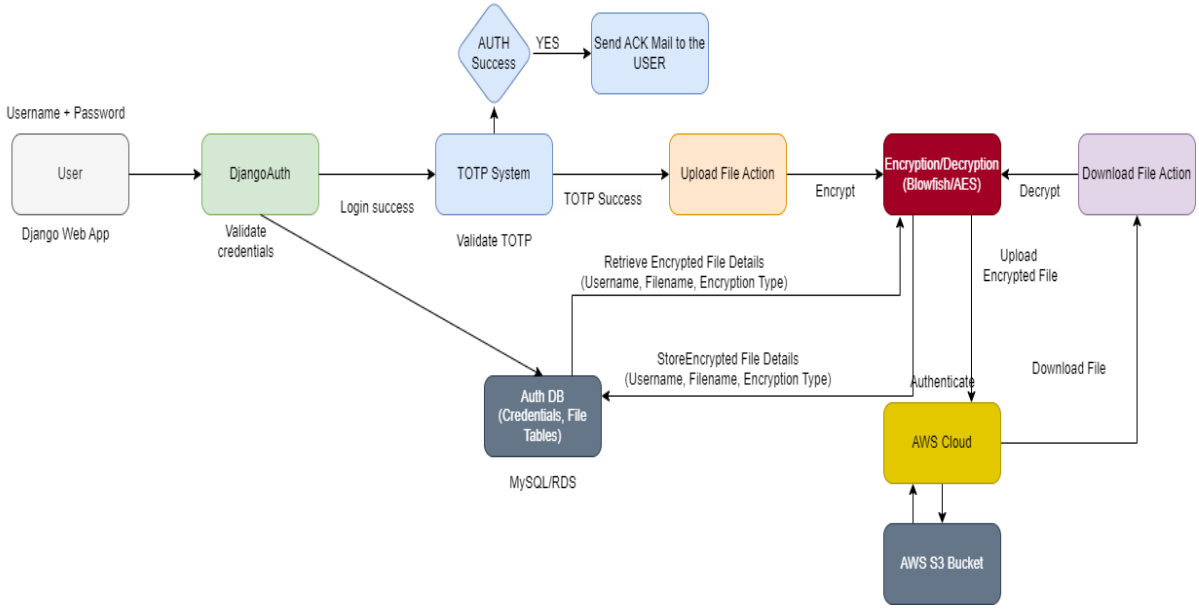

Figure 1: Proposed TOTP based 2FA Secure File Transfer Application

4 Design Specification

4.1 Proposed System Architecture

Figure 2 depicts the secure process for uploading and downloading files. It involves a client, a Django application, and an S3 bucket for storing files in the cloud. The process commences when the customer presents his login credentials to the Django application, which then authenticates the password. Two-factor authentication is implemented by utilizing a Time-based One-Time Password (TOTP) technique. After a successful authentication, an authenticated session is created. As an additional functionality a custom library has been developed to enhance security measures by collecting user IP addresses and geographical locations. This library integrates with the existing system and uses the SMTP protocol to send email notifications to users for additional verification.

When users logging in to their account, the custom library automatically gathers their IP address and location data. For the process of file upload, the client transmits the file that is to be uploaded together with a description of the file. The file is encrypted using the Blowfish/AES encryption techniques, resulting in an encrypted file. The encrypted file is stored using PUT object API operation in the S3 bucket. While downloading a file, the client initiates the process by requesting the file and providing its description. The encrypted file is obtained from the S3 bucket using the GET object API operation. The encrypted file is decoded using the appropriate algorithm used for encryption, Blowfish/AES decryption, resulting in the original decrypted file, which is then sent back to the client. The method employs many security mechanisms to guarantee the confidentiality and integrity of transferred files including password authentication, two-factor authentication (TOTP), file encryption/decryption utilizing robust algorithms (Blowfish/AES), and secure cloud storage (S3 bucket).

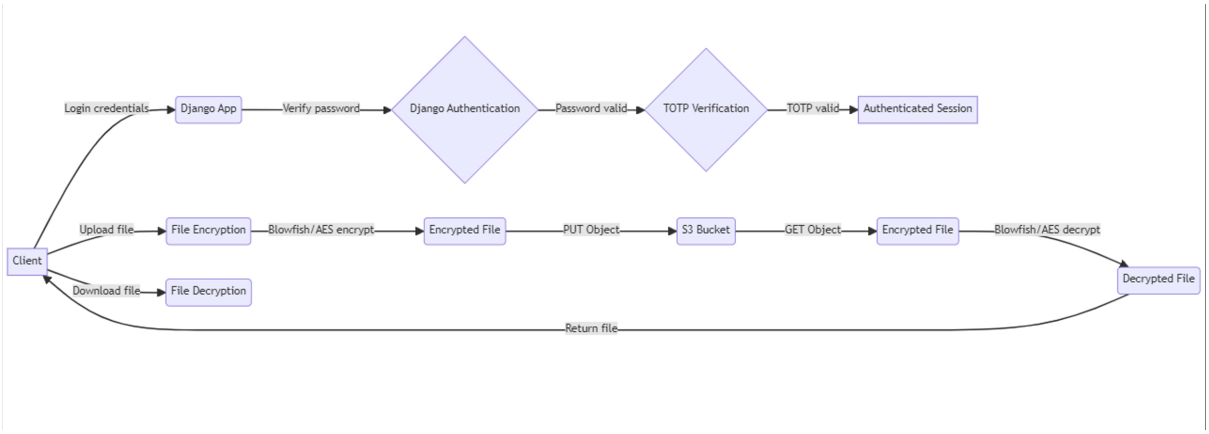

Figure 2: Proposed TOTP based 2FA Secure File Transfer Architecture

4.2 Blowfish and AES Encryption:

AES is founded on the design premise of a replacement permutation network. It utilizes a block size of 128 bits and offers key sizes of either 128, 192, or 256 bits.It processes a matrix of bytes called the state, which is organized in a column-major order with 4 columns. The majority of AES calculations are performed within a distinct finite field. The AES encryption is defined by a series of transformation rounds that iteratively turn the input plaintext into the final output of cipher text. The number of cycles of repetition is that there are 10 iterations of repetition for keys that are 128 bits in length. Twelve iterations of repetition for keys with a length of 192 bits, and approximately 14 iterations are required for 256-bit keys. The encryption process consists of four essential actions: SubBytes, ShiftRows, MixColumns, and XorRoundkey. Each round of encryption requires these operations to be performed. Decryption is the reverse process of encryption, accomplished by employing inverse operations such as InvSubBytes, InvShiftRows, and InvMixColumns.

Blowfish is a symmetric block cipher that operates on 64-bit blocks and supports variable-length keys. The algorithm consists of two components: a key expansion component and a data encryption component. The primary function of the key expansion component is to transform a key, which can be up to 448 bits in length, into many subkey arrays with a combined size of 4168 bytes. Data encryption is achieved via a 16-round Feistel network. It is only appropriate for use in situations where the key does not frequently change, such as a communications link or an automatic file encryption. When implemented on 32-bit microprocessors with big data caches, it exhibits notable speed advantages over the majority of encryption techniques. Encryption methods are designed in such a way that once a substantial amount of security analysis has been conducted, it is highly undesirable to modify the algorithm for the sake of efficiency. This would render the conclusions of the analysis invalid. Hence, it is crucial to concurrently consider security and performance during the design phase. Although it is not feasible to account for all potential computer architectures in the future, having knowledge of general optimization principles and conducting software experiments on current architectures to fine-tune performance can aid in achieving faster encryption algorithms in the future.

A comparison table is presented to compare AES and Blowfish algorithms in Table-1:

Table 1: Blowfish Vs AES - A Comparison

Parameter	Blowfish	AES
Key Length	448 bits	128, 192, 256 bits
Block Size	64	128
Speed	Faster	Fast
Security	Enough Security	Excellent Security
Structure	Feistel	Substitution Permutation
Key Cracking Time	3200 days	5x10power21 days

4.3 S3 Bucket Configuration

An S3 storage bucket will be created to store the encrypted files with the following configurations:

- Create an IAM user to limit access to AWS cloud computing platform with limited roles and access policies
- Restrict access to the bucket to only the Django app's IAM user via bucket policies
- Create specific buckets based on the user credentials.

The Django app will be assigned access to the IAM user with programmatic access. The access key will be securely stored in the app's configuration. Bucket policies will allow the IAM user PutObject and GetObject permissions on the bucket.

4.4 End-to-End Authentication Workflow

Figure 3 presents the sequence diagram for the end-to-end authentication flow in Django. The process begins when the client submits a login form to Django. Django then verifies the password with the Django Auth module. If the password is valid, Django prompts the client for a Time-based One-Time Password (TOTP). The client submits the TOTP, which Django Auth verifies. Upon successful TOTP verification, an authenticated session is established between the client and Django web application.

4.5 Sequence Diagram - Encryption Process

Figure 4 depicts a secure procedure for uploading and storing files as a sequence diagram. It involves a client, the Django web framework, encryption using the Blowfish/AES method, and an Amazon S3 bucket for storage. The process is initiated by the client when they upload a file to Django. Django employs Blowfish/AES encryption to encrypt the file. The encrypted file is then transferred to the S3 bucket for storage. After a successful upload, Django sends a confirmation to the client to indicate that the upload has been completed.

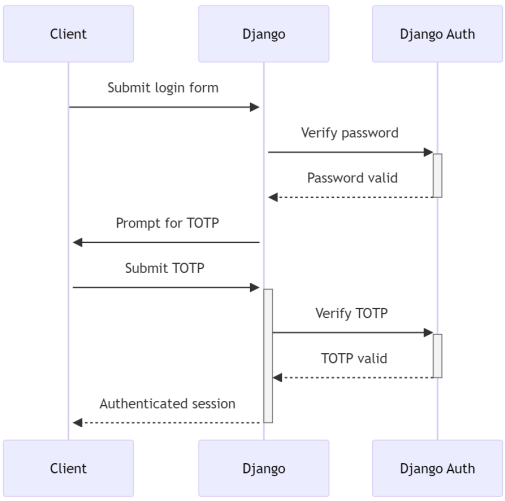

Figure 3: End-to-End Authentication Flow

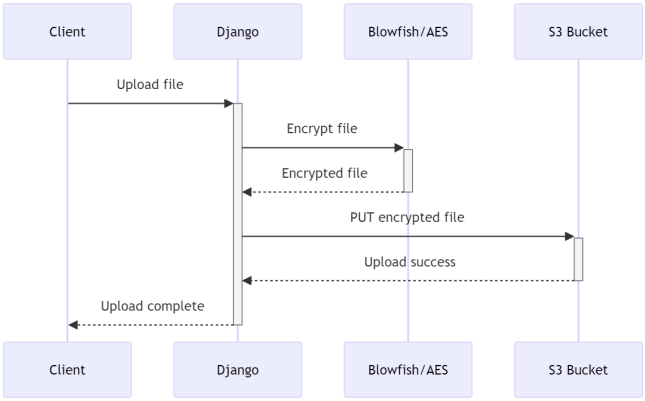

Figure 4: Encryption Flow

4.6 Sequence Diagram - Decryption Process

Figure 5 illustrates the secure file transmission process that includes a client, a Django web framework, an S3 cloud storage bucket, and encryption/decryption utilizing methods such as Blowfish or AES. The user triggers a request to download a file, which prompts Django to fetch the encrypted file from the S3 bucket. Django subsequently decrypts the file and delivers the decrypted rendition to the client.

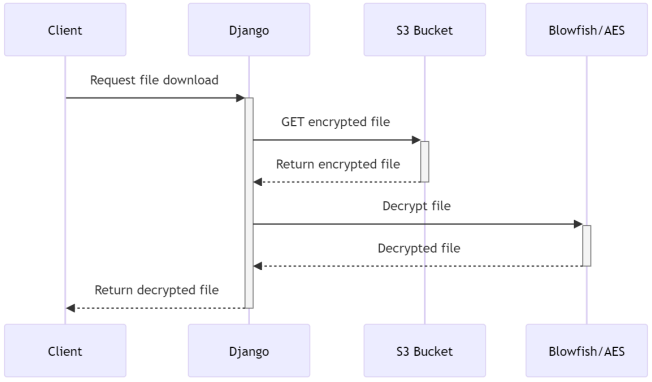

Figure 5: Decryption Flow

5 Implementation

5.1 Developing the Django Web Application

The initial step in the implementation process is to develop a Django web application that includes user registration, login, and file upload/download options. Create a new Django project and app using the ‘django-admin startproject‘ and ‘python manage.py startapp‘ commands, respectively. Define the necessary models for user registration and file management in the app’s ‘models.py‘ file. Create views and templates for user registration, login, and file upload/download functionality. The user registration view is implemented to handle user sign-up requests.

5.2 TOTP Authentication

Upon successful registration, the user is logged in and redirected to the appropriate page. To provide additional security of the Django application, a TOTP multi-factor authentication is implemented using the python library, pyotp, installed in the Django project environment. A specific view is created in views.py to handle the TOTP setup process, generate a secret key for the user and display a QR code that the user can scan using a TOTP app like Google Authenticator or Authy. the secret key is securely stored securely in the database, ’multistagecloud’, associated with the user’s account. The login view is modified to require TOTP authentication after successful password authentication.

5.3 Custom Library

A custom library was developed to collect the IP address and geographical location of users. It uses this information to automatically send email notifications to customers using the SMTP protocol and Google App passwords for authentication. This library is designed to keep customers informed about activity related to their accounts, such as logins and about file information.

5.4 Blowfish/AES Encryption and Decryption

The Blowfish and AES encryption and decryption modules are implemented as separate views using Python cryptography library, 'pycrypotodome'. When a user uploads a file, a random encryption key is generated using the Blowfish algorithm which is used to encrypt the file contents before saving them to S3 bucket storage. The S3 storage and bucket access is provided by the python library, boto3. After uploading, all the details related to the file metadata such as username, filename, encryption method and encryption key are stored securely in the 'multistagecloud' database. When a user requests to download an encrypted file, the corresponding encryption key is retrieved from the database to decrypt the file downloaded from the S3 storage bucket.

5.5 Django Web Application - Implementation

5.5.1 User Registration and TOTP Setup:

- The user visits the 'Register.html' page and enters their details like username, contact, email, address, and password.
- On submitting, the Signup view function is called, which generates a unique TOTP secret key using pyotp.random_base32().
- The user details, including the TOTP secret key, are stored in the 'register' table of the multistagecloud database, created using MySQL.
- A QR code is generated with the username and TOTP secret key, which can be scanned by the user with an authenticator app like Google Authenticator or Microsoft Authenticator to set up 2FA.

5.5.2 User Login and TOTP Verification:

- The user clicks the User.html page and enters their username and password for the login process.
- The 'UserLogin' view function verifies the provided credentials against the register table in 'multistagecloud' database.
- If the user credentials are valid, the user is asked to enter the TOTP code from their authenticator app on the LoginTOTP.html page.
- The LoginTOTPAction view function retrieves the user's TOTP secret key from the database and verifies the entered TOTP code using pyotp.TOTP(totp_secret).verify(totp_code).
- Upon successful verification, the user is granted access to the UserScreen.html page where the options to upload or download files are provided.

5.5.3 File Upload and Encryption:

- The user visits the UploadFile.html page and selects a file to upload and the encryption type (AES or Blowfish).
- The UploadFileAction view function is called upon form submission.

- If AES encryption is chosen, a random 256-bit key is generated using get_random_bytes(32), and the file is encrypted using AES.new(key, AES.MODE_CBC). The AES key is stored in the files table for later decryption.
- If Blowfish encryption is chosen, a fixed key is used (getCrowKey()), and the file is encrypted using Blowfish.new(getCrowKey(), mode=Blowfish.MODE_CBC).
- The encrypted file is uploaded to the AWS S3 bucket using the boto3 library.
- The file details, including the username, filename, encryption type, and AES key, when AES encryption was used, are stored in the 'files' table in 'multistagecloud' database.

5.5.4 File Download and Decryption:

- When the user visits the DownloadFile view function, a list of files associated with his/her username from the files table is retrieved.
- The user selects a file to download by clicking the corresponding link in the file table, which calls the DownloadFileAction view function with the filename and encryption type as parameters.
- The encrypted file is downloaded from the AWS S3 bucket using boto3.
- If the file is AES-encrypted, the corresponding AES key is retrieved from the files table, and the file is decrypted using AES.new(aes_key, AES.MODE_CBC).
- If the file is Blowfish-encrypted, it is decrypted using Blowfish.new(getCrowKey(), mode=Blowfish.MODE_CBC).
- The decrypted file is served to the user as an attachment for download.

The Django application uses the 'urls.py' file to associate URL patterns with the related view functions specified in 'views.py'. The program utilizes the 'pymysql' library to communicate with the MySQL database and the 'boto3' library to interface with AWS S3 to store and retrieve files.

5.6 Application Cloud Deployment

This project utilized Cloud9 which is a cloud-based Integrated Development Environment (IDE) provided by Amazon Web Services (AWS), for developing the application code. The codebase of the project was stored and managed using GitHub. For building and deploying the code, a CI/CD pipeline along with Elastic Beanstalk is used. In the first stage of the CI/CD pipeline, the source code is fetched from the GitHub and in the next stage the CI/CD pipeline automatically tests and builds the application code. Upon successful build, in the final stage of the pipeline, the code deploy deploys the application to the Elastic Beanstalk. If there are any changes made to your code of the github repository then automatically the CI/CD pipeline triggers and depolyes the updated application to the Elastic Beanstalk.

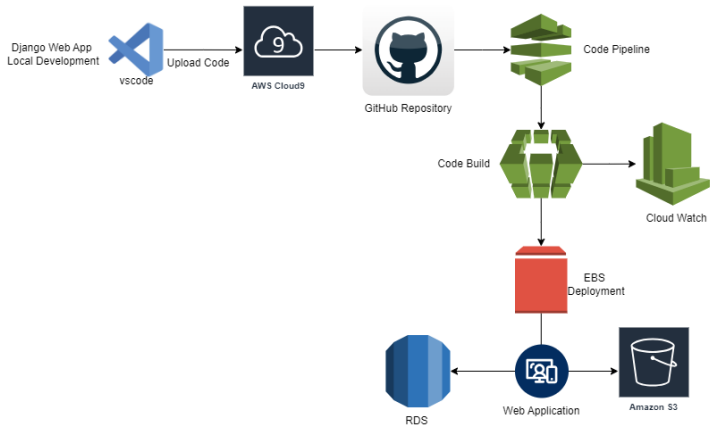

Figure 6: AWS Cloud Application Deployment

6 Evaluation

6.1 User Registration Process

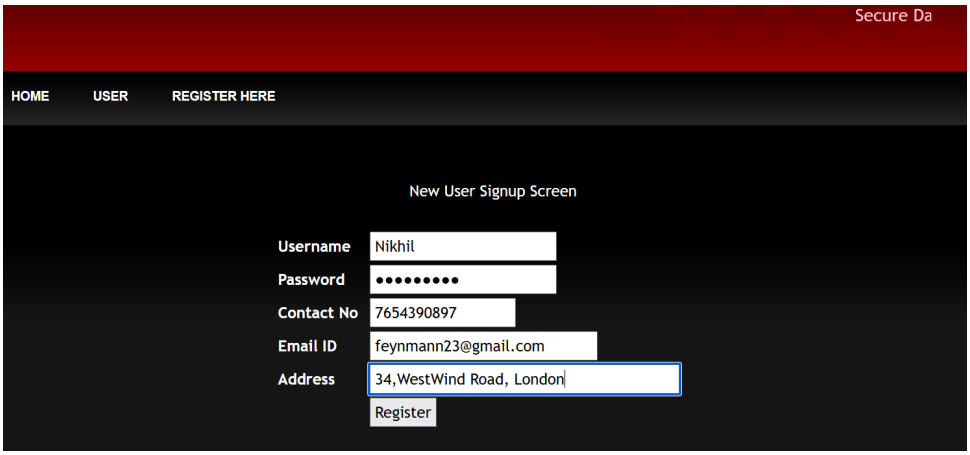

Figure 7: User Registration Process

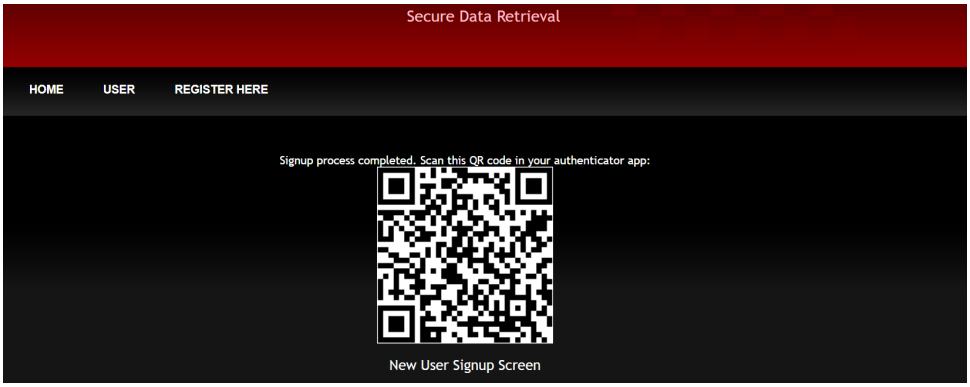

Figure 8: TOTP Registration Process

6.2 User Login with ToTP Authentication

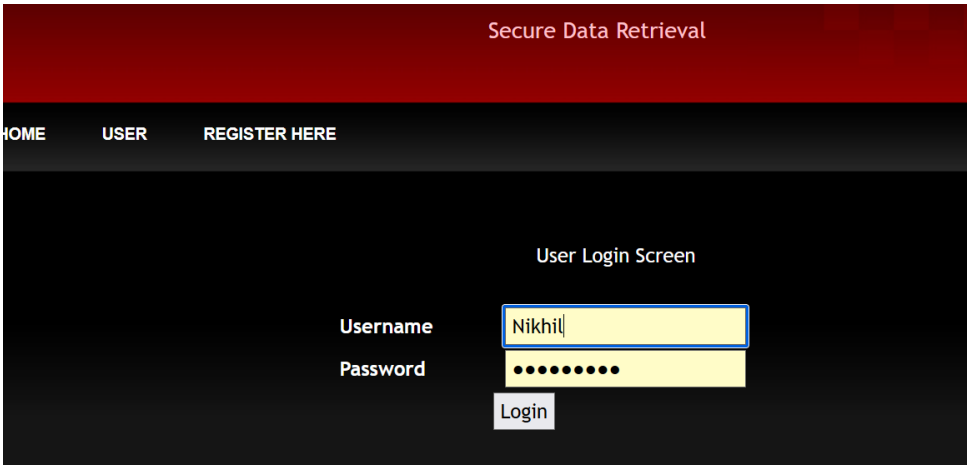

Figure 9: User Login

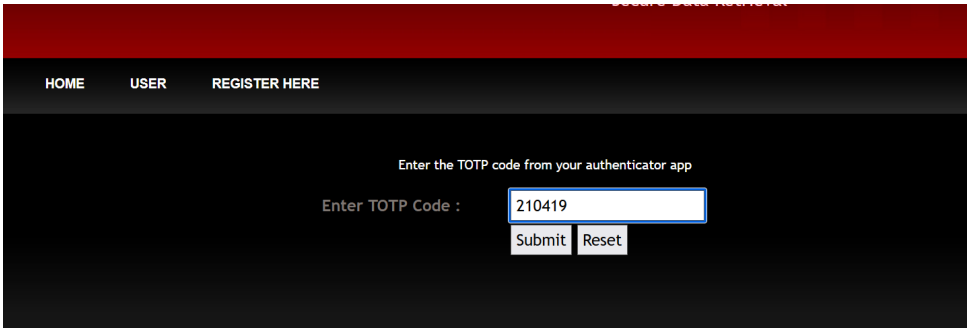

Figure 10: TOTP Verification

6.3 File Uploading after Encryption

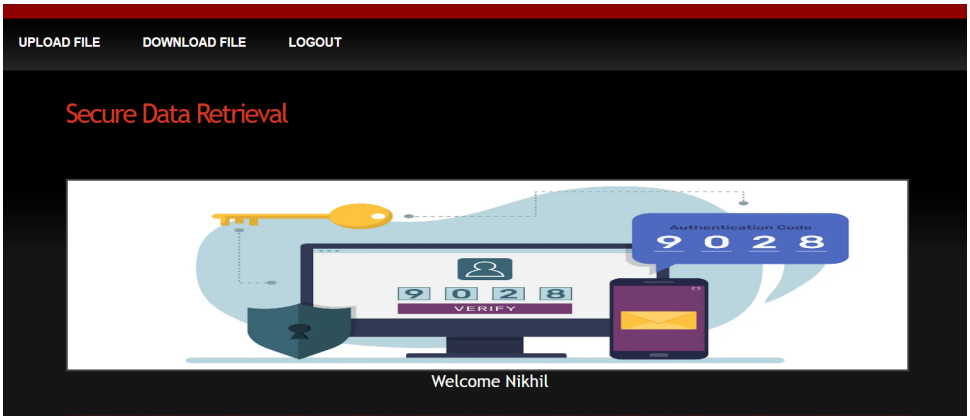

Figure 11: File Transfer Page

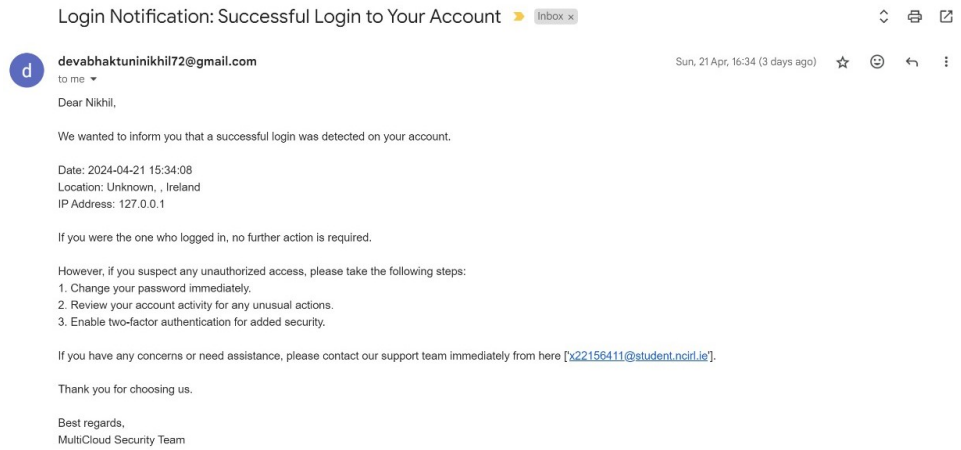

Figure 12: Notification mail sent to Customer

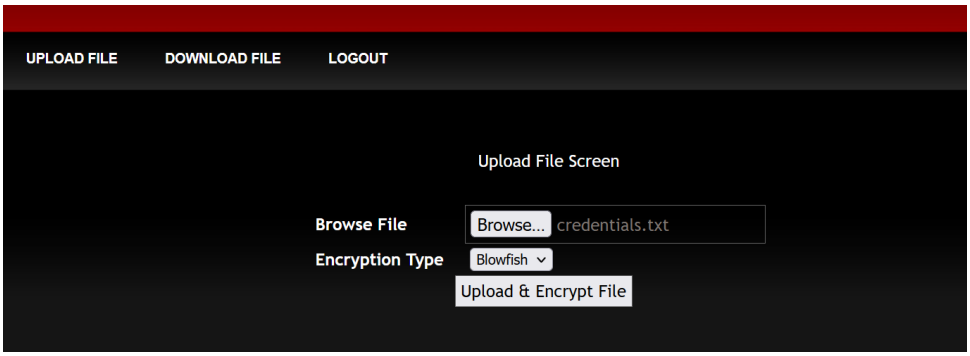

Figure 13: Blowfish Encrypted File Upload

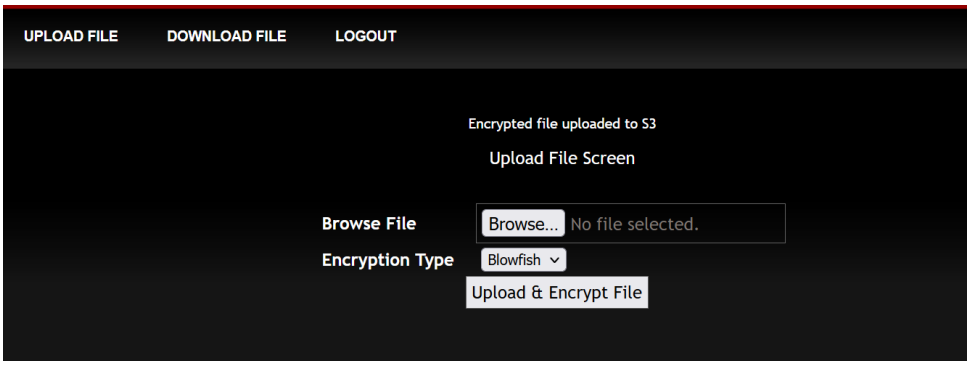

Figure 14: File Upload Completed - Blowfish

6.4 File Downloading after Decryption

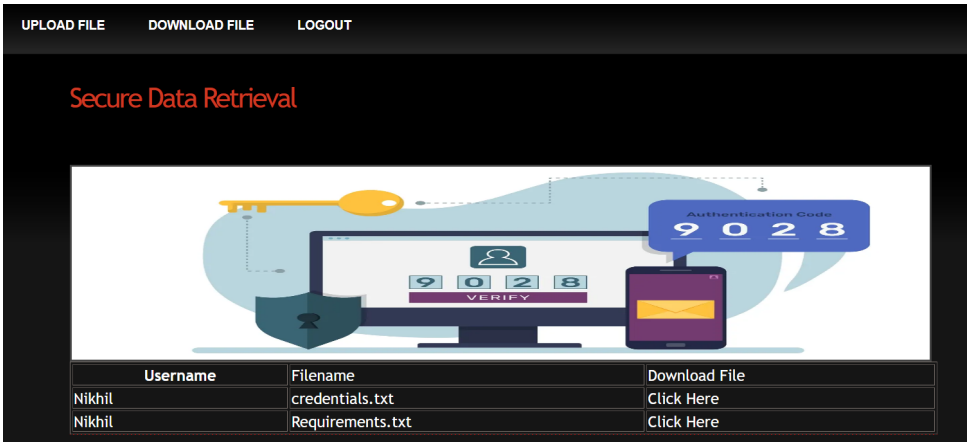

Figure 15: Download File Window

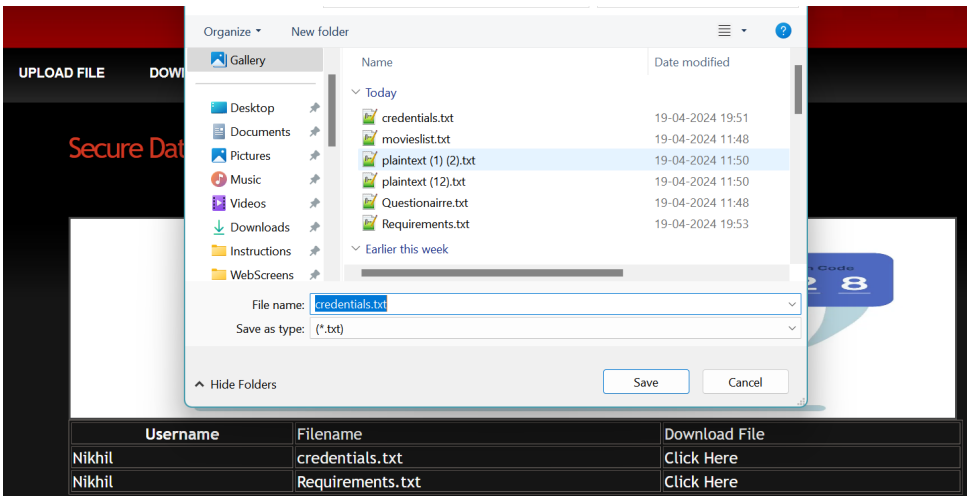

Figure 16: Save File - Blowfish

6.5 Discussion

The sections 6.1, 6.2, 6.3, 6.4 demonstrated the Django web application with registration process, login process, file uploading and downloading process from AWS S3 bucket. The performance of the AES and Blowfish algorithms were analysed by computing their encryption times and compression ratios for different file sizes. Here we have presented the analysis by computing the average encryption times and compression ratios of both AES and Blowfish encryption standards with text file sizes of 10KB, 20 KB, 50KB, 100KB, 200KB, 500KB and 1MB. This manner of examining the performance can provide a more constructive means of evaluation in determining the best algorithm. Table-1 showcases the average compression ratio and encryption times for both AES and Blowfish over the encryption of eight files with different file sizes and a plot describing their performance is presented in Figure 16. From Table-2, it can be seen that AES has significantly low encryption times compared to Blowfish suggesting that it would be the best algorithm

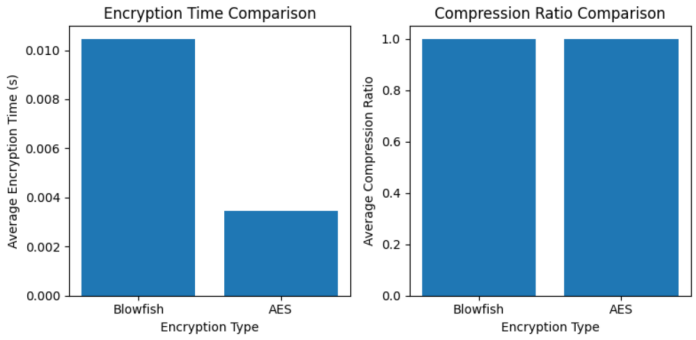

Figure 17: Encryption Metrics - AES and Blowfish

to use if the speed of the execution process is of high priority. The compression ratios of both AES and Blowfish are very similar with a slight advantage to AES. However, when transferring larger files of size in thousands of MB, the higher compression ratio can offer an excellent advantage. Hence, AES may be ideal algorithm for encryption when working with larger files, as this can save a lot on the cloud storage space and related costs.

Table 2: Evaluation Metrics - AES and Blowfish Encryption

Encryption type	Avg.Encryption Time (s)	Avg. Compression Ratio
Blowfish	0.010459	1.0003
AES	0.003444	1.0005

7 Conclusion and Future Work

This research was conducted to design a Django web application with multi-factored authentication system and secure file tranfer to AWS cloud storage by integrating TOTP authentication with the default Django authentication process and performing encryption using Blowfish and AES for securing file transfers to AWS S3 storage buckets. The inclusion of TOTP introduced an extra verification layer requiring users to enter a one-time password generated by an authenticator app,thereby increasing the security by preventing unauthorized access in case of user login information being compromised. The performance evaluation of Blowfish and AES encryption algorithms based on metrics like encryption times and compression ratios showed that AES outperformed Blowfish with better encryption times and slightly better compression ratios, making it the perfect candidate for scenarios prioritizing execution speed and when working with larger files to optimize cloud storage space and costs. The encrypted files are securely transferred to S3 buckets with strict access control permissions to prevent unauthorized access. The future scope of the research can involve automated key rotation for encryption keys, support for other cloud providers like Oracle, Microsoft or other third parties, conducting experiments with large files in sizes of gigabytes (GB), and risk-based adaptive authentication models that can adjust authentication policies based on risk profiles and user behavioral changes.

References

Acunetix (n.d.).
URL: *https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2020/*

Alouffi, B., Hasnain, M., Alharbi, A., Alosaimi, W., Alyami, H. and Ayaz, M. (2021). A systematic literature review on cloud computing security: threats and mitigation strategies, *IEEE Access* **9**: 57792–57807.

Duisebekova, K., Khabirov, R. and Zholzhan, A. (2021). Django as secure web-framework in practice, . . (1): 275–281.

Durga, K. K., Rejeti, V. K. K., Chandra, G. R. and Ramesh, R. (2023). Utilizing multi-stage authentication and an optimized blowfish algorithm for effective secure date retrieval on cloud computing, *Journal of Data Acquisition and Processing* **38**(4): 1418.

Ghimire, D. (2020). Comparative study on python web frameworks: Flask and django.

Gupta, I., Singh, A. K., Lee, C.-N. and Buyya, R. (2022). Secure data storage and sharing techniques for data protection in cloud environments: A systematic review, analysis, and future directions, *IEEE Access* .

Mashable (n.d.).
URL: *https://mashable.com/article/doordash-hack-customer-details-exposed*

Otta, S. P., Panda, S., Gupta, M. and Hota, C. (2023). A systematic survey of multi-factor authentication for cloud infrastructure, *Future Internet* **15**(4): 146.

Radhi, S. M. and Ogla, R. (2023). In-depth assessment of cryptographic algorithms namely des, 3des, aes, rsa, and blowfish, *Iraqi Journal of Computers, Communications, Control and Systems Engineering* **23**(3): 125–138.

Reddy, B. K. K. and Reddy, B. I. (2018). A comparative analysis of various multifactor authentication mechanisms, *International Journal of Scientific Research in Computer Science, Engineering and Information Technology* **3**(5): 8.

Reese, K., Smith, T., Dutson, J., Armknecht, J., Cameron, J. and Seamons, K. (2019). A usability study of five {two-factor} authentication methods, *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pp. 357–370.

Riaz, S., Khan, A. H., Haroon, M., Latif, S. and Bhatti, S. (2020). Big data security and privacy: Current challenges and future research perspective in cloud environment, *2020 International Conference on Information Management and Technology (ICIMTech)*, IEEE, pp. 977–982.

Seth, B., Dalal, S., Le, D.-N., Jaglan, V., Dahiya, N., Agrawal, A., Sharma, M. M., Prakash, D. and Verma, K. (2021). Secure cloud data storage system using hybrid paillier–blowfish algorithm., *Computers, Materials & Continua* **67**(1).

Singh, S., Singh, S. and Sharma, A. (n.d.). Real-time web-based secure chat application using django.

Singhal, V., Singh, D. and Gupta, S. (2022). Crypto stego techniques to secure data storage using des, dct, blowfish and lsb encryption algorithms, *Journal of Algebraic Statistics* **13**(3): 1162–1171.

Singhal, V., Singh, D. and Gupta, S. K. (2023). Data encryption technique based on enhancement of blowfish algorithm in comparison of des & dct methods, *Int. J. Sci. Res. in Computer Science and Engineering Vol* **11**(3).

Stigler, S. and Burdack, M. (2020). A practical approach of different programming techniques to implement a real-time application using django, *Athens J. Sci* **7**: 43–66.

Sun, P. (2020). Security and privacy protection in cloud computing: Discussions and challenges, *Journal of Network and Computer Applications* **160**: 102642.

Umarani, L. and Kumar, A. J. S. (n.d.). Multifactor authentication using double encryption based blowfish algorithm for data security in cloud environment.

Veeresh, V. and Parvathy, L. R. (2022). Data privacy in cloud computing, an implementation by django, a python-based free and open-source web framework, *International Journal of Intelligent Systems and Applications in Engineering* **10**(3s): 56–66.

Verizon (n.d.).
URL: *https://www.verizon.com/about/news/verizon-2021-data-breach-investigations-report*

Yang, P., Xiong, N. and Ren, J. (2020). Data security and privacy protection for cloud storage: A survey, *IEEE Access* **8**: 131723–131740.

Yu, X., Li, X., Wu, C. and Xu, G. (2023). Design and deployment of django-based housing information management system, *Journal of Physics: Conference Series*, Vol. 2425, IOP Publishing, p. 012018.