Multi-Factor Authentication System

Shreoshi Saharoy, Gorthi Lakshmi Sahitya, Anupama BS, Garapati Saranya
Department of Computer Science, *BITS*, Goa
h20240018@goa.bits-pilani.ac.in, h20240025@goa.bits-pilani.ac.in,
h20240043@goa.bits-pilani.ac.in, h20240136@goa.bits-pilani.ac.in

Group No. - 8

I. ABSTRACT

The Multi-Factor Authentication System is a secure web-based application that enhances user login protection by integrating an additional verification step using One-Time Passwords (OTP). Built with Django, the system supports OTP delivery via email or SMS and ensures sensitive data security using AES encryption. The project demonstrates a practical implementation of 2FA, offering a user-friendly interface, secure session handling, and modern design principles.

II. INTRODUCTION

In an era where digital threats are increasingly sophisticated, traditional password-based authentication is no longer sufficient to safeguard user data. Multi-Factor Authentication (MFA) has emerged as a reliable security approach that combines multiple verification methods to confirm a user's identity. This project aims to implement a robust Multi-Factor Authentication System using Django, enhancing security through OTP-based verification delivered via either email or SMS.

The system enables users to register with their credentials, choose their preferred OTP delivery method, and authenticate using both password and OTP. The backend is developed with Django, a high-level Python web framework, while the frontend is designed using Bootstrap for a modern and responsive user interface. The application also incorporates AES encryption to protect sensitive user data such as phone numbers, email addresses, and OTP secrets.

This project highlights the real-world application of cryptography, and best authentication practices, providing a secure and seamless login experience for end users.

III. TOOLS AND TECHNOLOGIES USED

The Multi-Factor Authentication System integrates multiple technologies across both frontend and backend to ensure secure, scalable, and user-friendly authentication. The following tools and frameworks have been used in the development:

A. Backend: Django Framework

- **Language:** Python
- **Framework:Django (v4.x)–** The core framework that handles routing, user authentication, database interaction, session management, and Cross-Site Request Forgery (CSRF) protection. Django's built-in features, such as User authentication, middleware, and form validation, streamline the development process while ensuring the application remains secure.
- **Password Hashing:** Django employs the PBKDF2 password hashing algorithm, which uses a salt to secure passwords. This prevents attackers from easily retrieving user passwords, even in the event of a database breach.

B. Frontend: HTML, CSS, JavaScript, and Bootstrap

- **HTML5 & CSS3:** The structure and styling of web pages are built using modern HTML5 and CSS3 features, ensuring compatibility across different browsers and devices.
- **JavaScript (Vanilla JS):** JavaScript is used for dynamic form handling, such as submitting login credentials or OTP through AJAX requests without reloading the page.
- **Bootstrap (v5.3):** A front-end framework that provides responsive design and modern UI components, enabling the system to be mobile-friendly and visually appealing.

C. Data Encryption: AES (Advanced Encryption Standard)

- Used for encrypting sensitive user data such as phone numbers and OTP secrets before storing them in the database.
- AES with a 256-bit key in CBC (Cipher Block Chaining) mode ensures high security.

D. Email Delivery: SMTP

- Django's built-in email backend is configured to send OTPs to registered user email addresses using an SMTP server (e.g., Gmail SMTP).
- Ensures users receive OTPs reliably via their email.

E. SMS Delivery: Twilio API

- Integrated with Twilio, a cloud communication platform, to send OTPs via SMS to Indian mobile numbers in +91 format.
- Twilio's secure API and global reliability make it suitable for production-grade 2FA systems.

F. Database: SQLite (default) / PostgreSQL (recommended for production)

- Stores user details, OTP preferences, and encrypted data.
- Django ORM ensures easy database queries, migrations, and model management.

G. Security Features

- **CSRF Tokens:** Built-in CSRF protection for all form submissions.
- **Session Management:** Handled by Django to track logged-in users.
- **Validation:** Input validation at both client and server ends to prevent malformed or malicious data.

IV. System Overview

The Multi-Factor Authentication System is a secure authentication platform that requires users to complete two steps to gain access to their accounts:

1) **Standard Login:** Users provide their username and password.
2) **OTP Verification:** After submitting valid credentials, an OTP is generated and sent to the user's registered email or phone number, depending on their preference.

The system is developed with a Django backend and a Bootstrap-based frontend. It uses AJAX-based form submissions for seamless interaction and provides secure user data handling through AES encryption. The OTP mechanism ensures time-sensitive verification, minimizing the risk of unauthorized access even in cases where a user's password is compromised. Key components of the system include:

- **User Registration:** Users sign up by providing a username, email, phone number (in +91 format), password, and preferred OTP delivery method.
- **Login and OTP Request:** After submitting valid login credentials, users receive a time-bound OTP through the selected method.
- **OTP Verification:** Users must enter the OTP correctly to gain access to their dashboard.
- **Session Management:** Django handles session authentication securely upon successful verification.
- **Encryption:** Sensitive fields like phone number and OTP secret are stored in encrypted format using AES (Advanced Encryption Standard).
- **Admin Monitoring:** The admin panel can manage the users that are registered and edit/delete their information as per requirement.

V. Architecture Design

The system follows a client-server model, where the frontend (client) communicates with the backend (Django server) via HTTP requests.

A. Frontend (Client)

- HTML templates for login, registration, and user dashboard.
- JavaScript handles form submissions and OTP input via AJAX.
- CSS via Bootstrap for a responsive UI.

B. Backend (Server - Django Framework)

- Authentication views
- OTP generation logic (time-bound)
- AES-based encryption module for sensitive data
- User model with extended fields for phone and OTP preferences
- Email/SMS integration (e.g., using SMTP for email and third-party API for SMS)

C. Database (SQLite/PostgreSQL)

- Stores user credentials, encrypted contact details, OTP secrets, and session information securely.

D. OTP Delivery Layer

- **Email:** Sent using Django's email backend.
- **SMS:** Sent using integrated SMS gateway API (Twilio).

VI. Implementation

The implementation of the Multi-Factor Authentication System involves the integration of a secure Django-based backend, a responsive frontend, and robust cryptographic mechanisms for safeguarding user information. This section outlines the technologies used and the step-by-step working of various modules.

A. Django Backend

The backend of the system is built using Django, a high-level Python web framework that promotes rapid development and clean design. Django handles User registration and authentication workflows; OTP generation and delivery; Secure storage and retrieval of sensitive data; Session management and access control.

B. Database

The system uses SQLite (default with Django) for data storage during development. It stores user records with fields including: Username, Hashed password, AES-encrypted email and phone number, OTP secret key, OTP delivery preferences (email or SMS).

C. AES Encryption of Sensitive Data

Sensitive user information such as email, phone number, and OTP secret key is encrypted using the Advanced Encryption Standard (AES) before being saved to the database. This ensures that even if the database is compromised, the information remains unreadable without the decryption key.

- AES is a symmetric-key algorithm that encrypts data using a 128-bit (or higher) key.
- The key is securely stored in Django's `settings.py` file and is not exposed in the codebase.
- Python's `cryptography` library is used for encryption and decryption.

Example Pseudocode:

```
from cryptography.fernet import Fernet
cipher = Fernet(settings.ENCRYPTION_KEY)
encrypted_email = cipher.encrypt(user_email.encode())
```

At the time of login or communication, the data is decrypted using the same key.

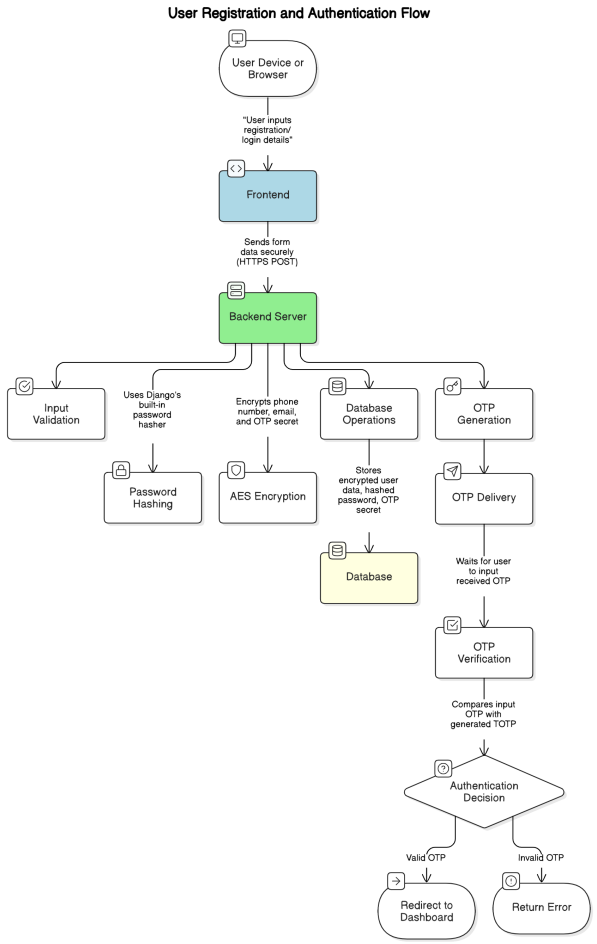

Fig. 1. System Flowchart

D. Password Hashing

Passwords are never stored in plaintext. Django's built-in password hashing mechanism uses the PBKDF2 algorithm with a SHA256 hash. When a user registers, their password is hashed and stored. A unique salt is added to each password before hashing to prevent attacks.On login, the entered password is hashed again and compared with the stored hash. Even if the database is exposed, the original password cannot be retrieved.Django manages hashing automatically with its User model and authenticate() method.

E. OTP Generation and Verification

The OTPs are generated using the Time-based One-Time Password (TOTP) algorithm. Each user is assigned a unique OTP secret key during registration, which is securely encrypted and stored in the database.

The system uses the Python library 'pyotp', which adheres to the TOTP standard defined in RFC 6238. Each registered user is assigned a unique OTP secret, a base32-encoded string securely stored in the database.

During the login flow, when an OTP is requested, the system generates it using the following logic:

```
totp = pyotp.TOTP(user.get_otp_secret())
otp = totp.now()
```

- `user.get_otp_secret()` retrieves the user's unique secret key.
- `pyotp.TOTP(...)` creates a time-based OTP generator object using that secret.
- `.now()` generates the current 6-digit OTP, valid for a 30-second window.

This ensures that each OTP is unique to the user and changes every 30 seconds. The short validity window enhances security by minimizing the chance of replay attacks.

F. Frontend Integration

The frontend is built using HTML5, CSS3 (Bootstrap 5), and JavaScript. It handles User input and form validation; Calling backend APIs via Fetch requests; OTP submission and response handling.Upon successful actions like registration or login, appropriate user feedback and redirection are provided dynamically. For example: On registration, the user is redirected to login. On login, the user is prompted to enter the OTPand once verified, access is granted to the dashboard. The UI is mobile-responsive and consistent with modern authentication platforms.

VII. Conclusion and Future Work

The Multi-Factor Authentication System developed in this project offers a secure and efficient way to protect user accounts by integrating multiple layers of authentication. By combining traditional password-based login with dynamic OTP verification through email or SMS, the system significantly reduces the risk of unauthorized access due to compromised credentials.The implementation leverages robust security practices, including: AES encryption, Password hashing, Time-based OTP generation, CSRF protection and session management .

The system is built using a modular and scalable architecture, making it adaptable for further enhancements such as biometric integration, push notifications, or integration with third-party authentication services (OAuth, SSO).This project not only demonstrates secure web development practices but also provides a solid foundation for real-world authentication systems that prioritize user privacy and data integrity.

References

[1] Twilio. *Twilio - Communication APIs for SMS, Voice, Video and Authentication*. https://www.twilio.com. Accessed April 14, 2025.

[2] Sasikumar, K., & Nagarajan, S. (2025). Enhancing Cloud Security: A Multi-Factor Authentication and Adaptive Cryptography Approach Using Machine Learning Techniques. *IEEE Open Journal of the Computer Society*.

[3] Chong, C. L., & Harun, N. Z. (2025). Secure File Sharing System with Strong Password and One Time Password Authentication. *Journal of Computing Research and Innovation*, *10*(1), 98–107.

[4] Choudhary, P., Das, S., Potta, M. P., Das, P., & Bichhawat, A. (2025). *Online Authentication Habits of Indian Users*. arXiv preprint arXiv:2501.14330.

[5] Chinta, P. C. R., Moore, C. S., Karaka, L. M., Sakuru, M., Bodepudi, V., & Maka, S. R. (2025). *Building an Intelligent Phishing Email Detection System Using Machine Learning and Feature Engineering*. European Journal of Applied Science, Engineering and Technology, 3(2), 41–54.

[6] Han, J. (2024). CNN-based multi-factor authentication system for mobile devices using faces and passwords. *Applied Sciences*, *14*(12), 5019. https://doi.org/10.3390/app14125019.

[7] Kumar, R., & Karthikeyan, K. (2023). *Improvised Multi-Factor Authentication for End-User Security in Cyber-Physical Systems*. International Journal of Intelligent Systems and Applications in Engineering, 11(1), 123–130.

[8] Suleski, T., Ahmed, M., Yang, W., & Wang, E. (2023). *A Review of Multi-Factor Authentication in the Internet of Healthcare Things*. Digital Health, 9, 20552076231177144. https://doi.org/10.1177/20552076231177144

[9] Shewale, C., Kadam, S., Shelke, P., Patil, R., Dedgaonkar, S., & Banubakode, A. (2023). *Multi-Factor Authentication and IP Address Restriction Based Question Paper Delivery System for Indian Universities*. International Journal on Recent and Innovation Trends in Computing and Communication, 11(10), 390–396.

[10] Prabakaran, D., & Ramachandran, S. (2022). *Multi-Factor Authentication for Secured Financial Transactions in Cloud Environment*. Computers, Materials & Continua, 70(1), 1781–1798. https://doi.org/10.32604/cmc.2022.019591

[11] Iyanda, A. R., & Mayokun, E. F. (2022). *Development of two-factor authentication login system using dynamic password with SMS verification*. International Journal of Education and Management Engineering, 12(3), 13.

[12] Mhatre, D. D., & Nag, S. (2022). *Study of Multi-Factor Authentication*. International Journal of Advanced Research in Science, Communication and Technology, 2(2), 45–50.

[13] Garg, S., Batra, J., Sachdeva, M., & Chauhan, A. (2021). *Multi-Factor Authentication for Secured Financial Transactions in Cloud Environment*. Computers, Materials & Continua, 70(1), 657–671. https://doi.org/10.32604/cmc.2022.019798

[14] Jacomme, C., & Kremer, S. (2021). *An extensive formal analysis of multi-factor authentication protocols*. ACM Transactions on Privacy and Security (TOPS), 24(2), 1–34.

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