Text Encryption Web Application

Prepared By: Surya Hanuman KONJETI

Position: Cyber Security Intern | Intern ID: SMI81626

College: ESAIP, École d'Ingénieurs

Company: Slash Mark IT Solutions (OPC) Pvt. Ltd.

Batch: November 15, 2025 to January 15, 2026

Index

1. Project Overview

This project implements a secure Text Encryption and Decryption Web Application using Python, Flask, and the Cryptography library. It provides a simple user interface where users can enter plaintext and generate encrypted tokens using AES-GCM encryption. Decryption is performed using the same password, ensuring confidentiality and integrity. The system uses modern cryptographic standards including PBKDF2-based key derivation, random salt, and random nonce generation.

2. Objectives

• Implement strong symmetric encryption using AES-GCM.

• Understand and use PBKDF2 for password-based key derivation.

• Ensure randomness using salt and nonce for secure encrypted output.

• Build a user-friendly Flask-based encryption web application.

• Demonstrate secure handling of user data during encryption/decryption.

3. Tools & Technologies

• Python 3 – core programming language.

• Flask – lightweight backend framework for web UI.

• Cryptography Library – AES-GCM encryption and PBKDF2 key derivation.

• VS Code – development environment.

• Browser (Chrome/Edge) – running the web application.

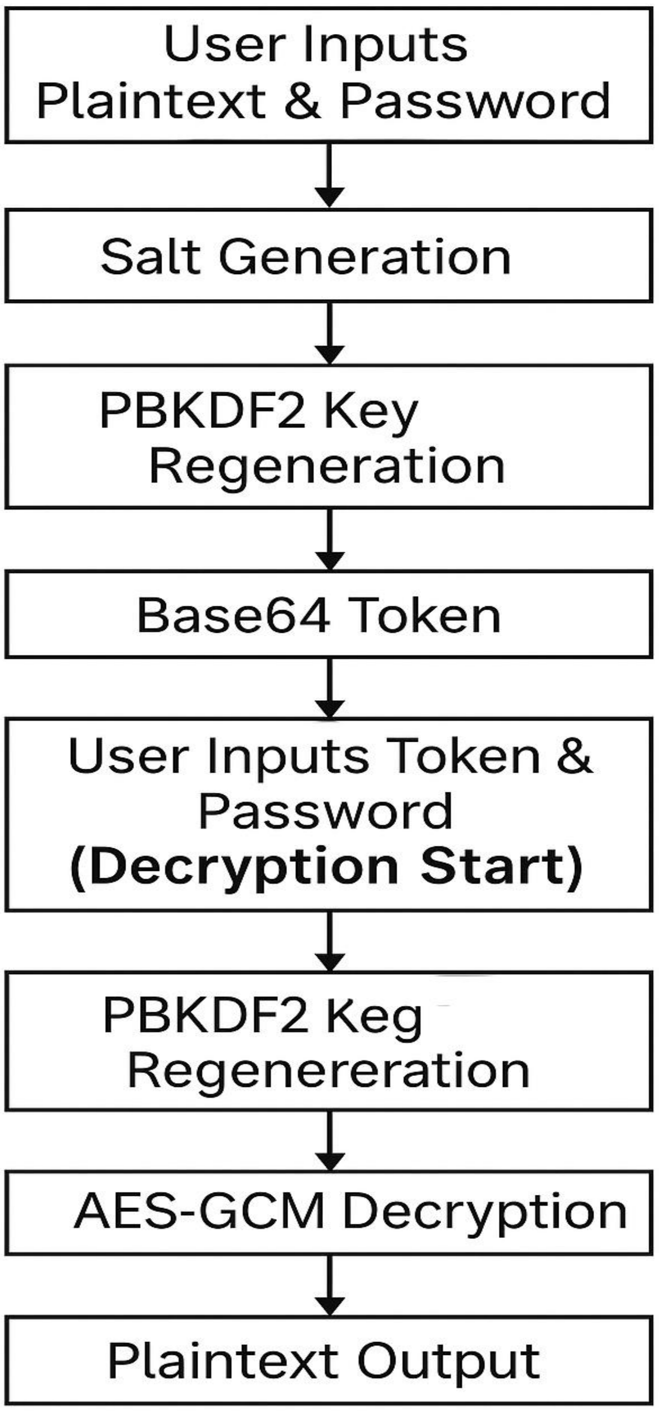
• Virtual Environment (venv) – isolated Python environment.

4. System Architecture

The system is based on secure cryptographic workflow principles. The encryption process begins when the user inputs plaintext and password. The application generates a random salt and derives a secure AES-256 key using PBKDF2. A unique nonce is created for AES-GCM encryption, ensuring that identical plaintext never produces identical ciphertext. The final encrypted output is packaged as a Base64 token containing salt, nonce, and ciphertext.

During decryption, the token is decoded, and salt, nonce, and ciphertext are extracted. Using the same password and salt, PBKDF2 regenerates the AES key. AES-GCM decrypts the ciphertext and validates integrity. If any component is modified (salt, nonce, ciphertext, password), decryption fails.

5. Architecture Diagram


User Inputs Plaintext & Password
Salt Generation
PBKDF2 Key Regeneration
Base64 Token
User Inputs Token & Password (Decryption Start)
PBKDF2 Keg Regenereration
AES-GCM Decryption
Plaintext Output

Detailed Explanation of Architecture Diagram

User Inputs Plaintext & Password
The user starts the process by entering the plaintext message they want to encrypt along with a password. The password is the foundation for deriving the encryption key.

Salt Generation
A random salt (16 bytes) is generated to add randomness to the password. This ensures that even if the same password is used multiple times, the derived key is always unique.

PBKDF2 Key Regeneration (Encryption)
PBKDF2 (Password-Based Key Derivation Function 2) uses the password and salt to generate a cryptographically strong 256-bit AES key. It runs thousands of hashing iterations to make brute-force attacks extremely difficult.

AES-GCM Encryption
AES-GCM (Advanced Encryption Standard - Galois Counter Mode) encrypts the plaintext using the derived key. It also generates authentication tags that ensure integrity and protect against tampering.

Base64 Token Output
After encryption, the salt, nonce, and ciphertext are combined and encoded in Base64 to generate a single encrypted token. This token contains everything needed for decryption.

User Inputs Token & Password (Decryption Start)
To decrypt, the user enters the encrypted token and the same password used during encryption. The process will fail if even a single character or password mismatch occurs.

Token Decoding
The Base64 token is decoded to extract the original raw components: salt, nonce, and ciphertext. These components are essential to reconstruct the AES key and decrypt the message.

PBKDF2 Key Regeneration (Decryption)
Using the extracted salt and the user-provided password, PBKDF2 regenerates the same AES key that was used during encryption. Only the correct password leads to the correct key.

AES-GCM Decryption
AES-GCM uses the regenerated key and nonce to decrypt the ciphertext. It also checks integrity. If the ciphertext was modified or password is wrong, decryption fails immediately.

Plaintext Output
If all components are correct and authentication matches, the system successfully retrieves the original plaintext message and displays it to the user.

6. Code Explanation

The project consists of two main Python files:
1. crypto_app.py – Handles the cryptographic logic including PBKDF2 key derivation, salt generation, nonce creation, AES-GCM encryption, and decryption.
2. app.py – Flask backend that displays the web interface, accepts user input, and calls crypto_app functions to perform encryption/decryption.

7. Web Application UI

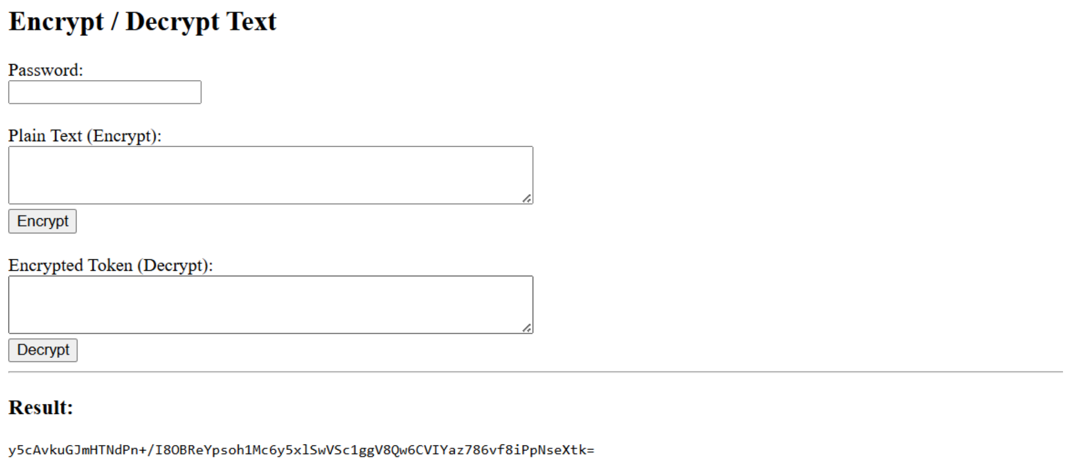
The UI contains three input areas:
• Password field
• Plaintext field
• Encrypted token field
Users can encrypt plaintext or decrypt tokens easily using the provided buttons.

8. Testing Screenshots

Encryption and Decryption Outputs

This section presents the actual outputs generated by the application during encryption and decryption tests. These outputs validate that the system functions correctly and securely using AES-GCM encryption and PBKDF2 key derivation.

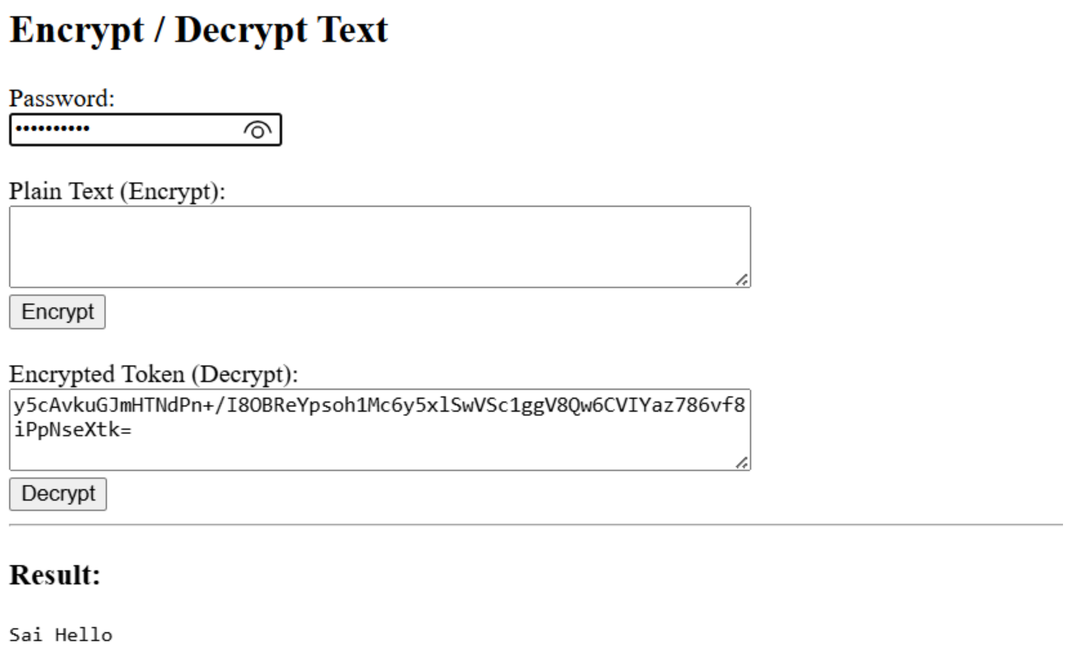
8.1 Encryption Output



```
y5cAvkuGJmHTNdPn+/I8OBReYpsoh1Mc6y5xlSwVSc1ggV8Qw6CVIYaz786vf8iPpNseXtk=
```

The above screenshot shows the encryption result. When the user enters plaintext and a password, the system generates a secure encrypted token. This token contains the salt, nonce, and ciphertext encoded in Base64. Even if the same plaintext and password are used again, a different encrypted token is produced due to the randomness of salt and nonce.

8.2 Decryption Output



Encrypt / Decrypt Text

Password:
••••••••••

Plain Text (Encrypt):

Encrypt

Encrypted Token (Decrypt):

```
y5cAvkuGJmHTNdPn+/I8OBReYpsoh1Mc6y5xlSwVSc1ggV8Qw6CVIYaz786vf8
iPpNseXtk=
```

Decrypt

Result:

```
Sai Hello
```

The above screenshot shows the decryption result. When the correct password and token are entered, the system successfully extracts the salt, nonce, and ciphertext to regenerate the AES key using PBKDF2 and reverses AES-GCM encryption to reveal the original plaintext. If the password is incorrect or the token is modified, decryption will fail — ensuring data integrity.

9. Conclusion

This project successfully demonstrates modern encryption techniques using AES-GCM and PBKDF2. The implementation ensures high security by using random salt, random nonce, and strong key derivation. The web interface makes the encryption system accessible and practical for real world use cases. This project also builds foundational knowledge required in cybersecurity, cryptography, and secure application development.