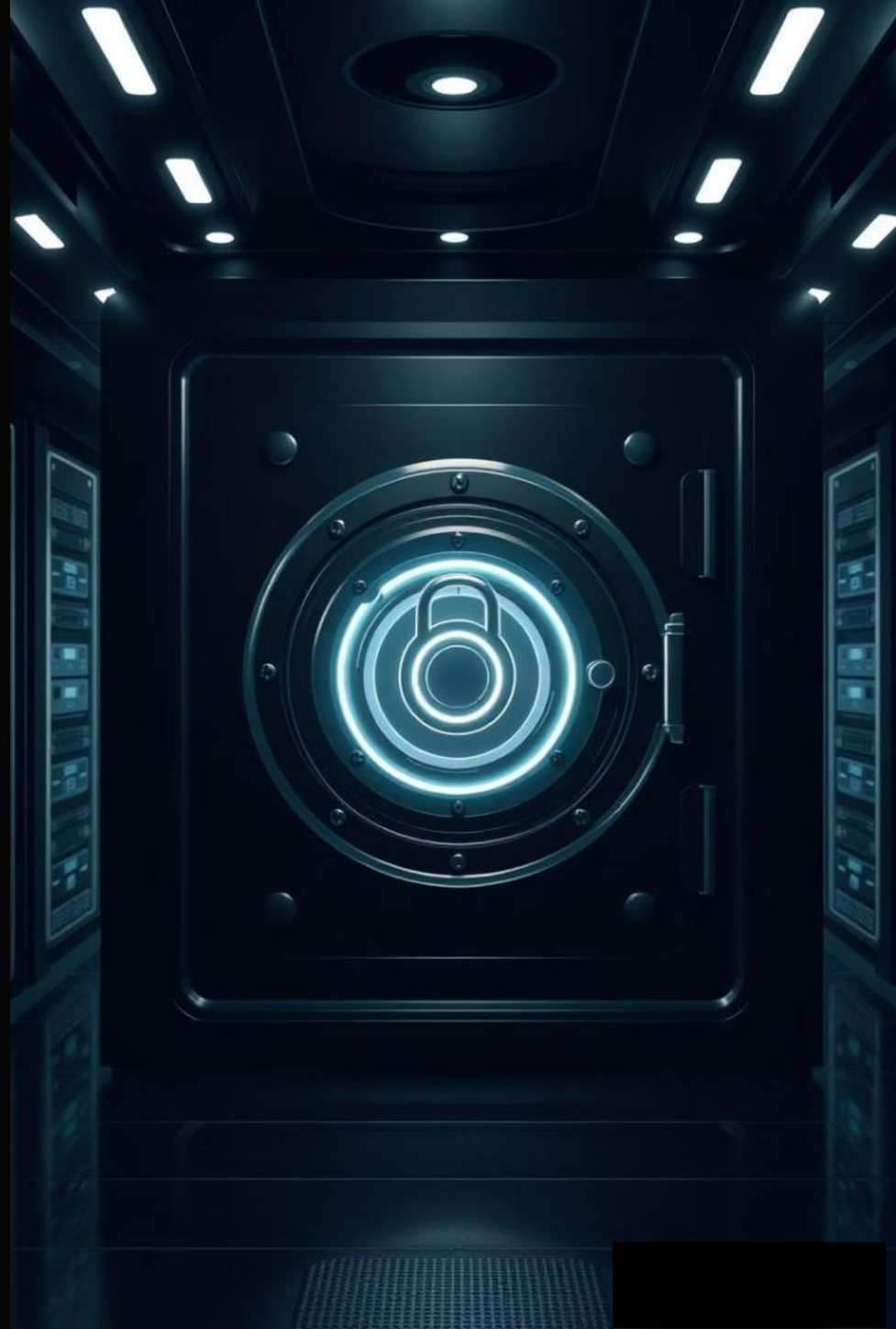
Building a Simple Cryptography Library

Embark on a journey to understand basic cryptography. Learn to create a small encryption tool. This is for educational purposes, not production use.

Disclaimer: This library is not for sensitive data. It's an example only.



Symmetric vs. Asymmetric Encryption

Symmetric Encryption

Uses a single secret key for encryption and decryption.

- Faster, for bulk data
- Examples: AES, DES

This library focuses on symmetric encryption for simplicity.

Asymmetric Encryption

Uses key pairs: public (encrypt) and private (decrypt).

- Slower, for key exchange
- Example: RSA

Choosing AES for Encryption

AES Standard

AES is a widely used, secure symmetric encryption algorithm.

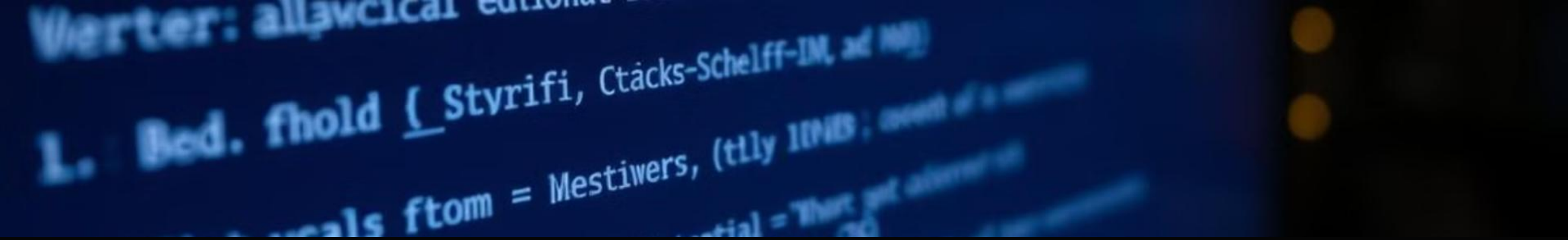
128-bit Key

Key sizes include 128, 192, or 256-bit. This example uses 128-bit.

16-byte Block

The block size is 128 bits (16 bytes).

AES is supported across programming languages. It's relatively easy to implement.



Library Structure: Key and IV Generation

Key Generation

Generate a 128-bit random key securely.

CSPRNG

Use a cryptographically secure random number generator.

Key Storage

Store the key securely.

Initialization Vector

Use IV with same length as AES block size.

Library Structure: Decryption Steps

Retrieve

Get the original key and IV.

Any error in the key or IV results in incorrect decryption.

Decrypt

Use AES in decryption mode.

Reverse

Function reverses the process.

Sample Python Code

Here is a conceptual example of the library written in Python.

```
from Crypto.Cipher import AES
import secrets

key = secrets.token_bytes(16)
cipher = AES.new(key, AES.MODE_EAX)
nonce = cipher.nonce
ciphertext, tag = cipher.encrypt_and_digest(plaintext.encode('ascii'))

cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
plaintext = cipher.decrypt(ciphertext).decode('ascii')
```

CRYPTOGRAPHY AND NETWORK SECURITY

Project Report

Build a Simple Cryptography Library for Encrypting Messages

Bachelor of Computer Application

PROJECT GUIDE: **Mr. Narayan Vyas**

SUBMITTED BY: **Rohit Kumar Singh (23CSA2BC261).**

March 2025

VIVEKANANDA GLOBAL UNIVERSITY

ACKNOWLEDGEMENT

*I have taken this opportunity to express my gratitude and humble regards to the Vivekananda Global University to provide an opportunity to present a project on the "*Build a Simple Cryptography Library for Encrypting Messages*" which is a database security fundamentals based project.*

*I would also be thankful to my project guide "**Mr.** **Narayan Vyas"** to help me in the completion of my project and the documentation.I have taken efforts in this project but the success of this project would not be possible without his support and encouragement.*

*I would like to thanks our Dean sir **"Dr.Surendra Yadav"** and Hod sir **"Dr.Rakesh Sharm"** to help us in providing all the necessary books and other stuffs as and when required .I show my gratitude to the authors whose books has been proved as the guide in the completion of my project I am also thankful to my classmates and friends who have encouraged me in the course of completion of the project.*

Thanks

Rohit Kumar Singh (23CSA2BC261)

Place: Jaipur

Date:

DECLARATION

We hereby declare that this Project Report titled “Build a Simple Cryptography Library for Encrypting Messages” submitted by us and approved by our project guide, to the Vivekananda Global University, Jaipur is a bonafide work undertaken by us and it is not submitted to any other University or Institution for the award of any degree diploma / certificate or published any time before.

Project Group

Student Name: **Rohit Kumar Singh (23CSA2BC261)**

Project Guide: **Mr. Narayan Vyas**

Table of Contents

1 Project Title..... 5

2 Problem Statement..... 5

3 Project Description..... 5

3.1 Scope of the Work..... 5

3.2 Project Modules..... 5

3.3 Context Diagram (High Level)..... 5-6

4 Implementation Methodology..... 6

5 Technologies to be used..... 6

5.1 Software Platform..... 6

5.2 Hardware Platform..... 6

5.3 Tools..... 6

6 Advantages of this Project..... 6-7

7 Assumptions, if any..... 7

8 Future Scope and further enhancement of the Project..... 7

9 Definitions, Acronyms, and Abbreviations..... 7

10 Conclusion..... 7

11 References..... 7

Appendix..... 8

A: Screen Shots..... 8-12

Project Title:

"Secure File Encryption & Decryption Using AES"

1. Problem Statement:

In the digital world, securing sensitive files is crucial to prevent unauthorized access, data breaches, and cyber threats. This project aims to develop a **secure file encryption and decryption system** using AES (Advanced Encryption Standard).

2. Project Description:

This project allows users to encrypt files securely and decrypt them when needed. The AES-256 encryption standard ensures data integrity and security.

3.1 Scope of Work:

- Encrypt and decrypt any file (e.g., `.txt`, `.pdf`, `.jpg`, etc.).
- Generate a **random AES key** for encryption.
- Ensure the key is securely stored for decryption.
- User-friendly command-line interface.

3.2 Project Modules:

1. **Key Generation Module** – Generates and saves a secure key.
2. **Encryption Module** – Encrypts a file using AES encryption.
3. **Decryption Module** – Decrypts an encrypted file.
4. **File Handling Module** – Handles file read/write operations.

3.3 Context Diagram (High Level):

- User uploads a file → System encrypts the file → Stores encrypted file → User decrypts the file when needed.

4. Implementation Methodology:

We will use Python's `cryptography.fernet` module to handle AES encryption and decryption.

5. Technologies to be Used:

5.1 Software Platform:

- Python 3.x
- Cryptography Library (`pip install cryptography`)

5.2 Hardware Platform:

- Any standard PC/Laptop

5.3 Tools:

- Python
- Terminal/Command Prompt

6. Advantages of this Project:

✓ Secure and reliable file encryption using AES.
✓ Prevents unauthorized access to sensitive data.
✓ Can be extended for cloud storage encryption.

7. Assumptions:

- Users will securely store their encryption keys.
- Python 3.x is installed on the system.

8. Future Scope:

Enhance security by implementing **multi-factor authentication** (MFA).
Integrate GUI for better user experience.
Implement **hybrid encryption** (AES + RSA) for improved security.

9. Definitions, Acronyms, and Abbreviations:

- **AES:** Advanced Encryption Standard
- **Encryption:** Converting data into a secret format
- **Decryption:** Converting encrypted data back to its original form

10. Conclusion:

This project provides a robust file encryption system, ensuring data privacy and security.

11. References:

- **AES Encryption** - NIST Documentation

Python Code Implementation:

Here is the complete Python code for **File Encryption & Decryption using AES**:

```
from cryptography.fernet import Fernet
import os

class FileEncryptor:
    def __init__(self, key_file='encryption_key.key'):
        """Initialize the encryption system."""
        self.key_file = key_file
        self.key = self.load_or_generate_key()

    def load_or_generate_key(self):
        """Loads an existing key or generates a new one if not found."""
        if os.path.exists(self.key_file):
            with open(self.key_file, 'rb') as file:
                return file.read()
        else:
            key = Fernet.generate_key()
            with open(self.key_file, 'wb') as file:
                file.write(key)
            return key

    def encrypt_file(self, file_path):
        """Encrypt a file using AES encryption."""
        fernet = Fernet(self.key)
        with open(file_path, 'rb') as file:
            file_data = file.read()

        encrypted_data = fernet.encrypt(file_data)

        encrypted_file_path = file_path + '.enc'
        with open(encrypted_file_path, 'wb') as file:
            file.write(encrypted_data)

        print(f"File '{file_path}' encrypted successfully as
'{encrypted_file_path}'")
        return encrypted_file_path

    def decrypt_file(self, encrypted_file_path):
        """Decrypt a file using AES decryption."""
        fernet = Fernet(self.key)
        with open(encrypted_file_path, 'rb') as file:
            encrypted_data = file.read()

        decrypted_data = fernet.decrypt(encrypted_data)

        original_file_path = encrypted_file_path.replace('.enc', '')
        with open(original_file_path, 'wb') as file:
            file.write(decrypted_data)

        print(f"File '{encrypted_file_path}' decrypted successfully as
'{original_file_path}'")
        return original_file_path
```

```
# Example Usage
if __name___ == "__main__":
    encryptor = FileEncryptor()

    file_to_encrypt = "example.txt"  # Change this to your file name
    encrypted_file = encryptor.encrypt_file(file_to_encrypt)

    decrypt_choice = input("Do you want to decrypt the file? (yes/no): ").lower()
    if decrypt_choice == "yes":
        encryptor.decrypt_file(encrypted_file)
```

How to Run the Project?

1. Install dependencies:
2. `pip install cryptography`
3. Create a sample text file:
4. `echo "This is a secret message" > example.txt`
5. Run the script:
6. `python file_encryptor.py`
7. Follow the prompts to encrypt/decrypt files.

A: Screenshots (Expected Output)

1. **Encryption Success:**
2. `File 'example.txt' encrypted successfully as 'example.txt.enc'`
3. **Decryption Success:**
4. `File 'example.txt.enc' decrypted successfully as 'example.txt'`

Why Choose This Project?

✓ **Practical Application** – Useful for securing sensitive files.
✓ **Easy Implementation** – Uses Python's cryptography module.
✓ **Can be Extended** – Can be integrated with cloud storage or databases.