CRYPTOACOUSTIX: BLOCKCHAIN-DRIVEN AUDIO ENCRYPTION FRAMEWORK FOR DATA INTEGRITY AND CONFIDENTIALITY IN CLOUD SYSTEMS

[1]Muthusamy P, [2]Sathiyapriya P, [3]Deepika S, [4]Priyadharshini L, [5]Ramya C

[1]Professor, Department of CSE(Cyber Security), Muthayammal Engineering College.

[2]Assistant Professor, Department of CSE(Cyber Security), Muthayammal Engineering College.

[3,4,5]Student, Department of CSE(Cyber Security), Muthayammal Engineering College.

[1]mpmmuthu6@gmail.com, [2]sathiyapriya18mec@gmail.com, [3]d9190447@gmail.com, [4]priyachandru1465@gmail.com, [5]r7683570@gmail.com

ABSTRACT - In the modern era of digital transformation, cloud storage has become the backbone of data management, but it also introduces critical security challenges involving confidentiality, integrity, and authenticity. The system takes security a step further by embedding encryption keys and XOR-generated codes within audio files using the Least Significant Bit (LSB) steganography technique, concealing them from detection. This innovative approach not only safeguards sensitive data stored in the cloud but also leverages decentralized trust and imperceptible data hiding techniques to ensure resilience against modern cyber threats. These stego-audio files are then encrypted using Elliptic Curve Cryptography (ECC), providing multi-layered protection against brute-force and data interception attacks. The integration of AI and ML for dynamic threat analysis and intelligent key management, paving the way for a self-adaptive, intelligent, and highly secure cloud data protection ecosystem.

KEYWORDS - Blockchain, Audio Encryption, Data Integrity, Confidentiality, Cloud Systems, Cryptography, and Security

1. INTRODUCTION

The exponential growth of cloud computing has revolutionized the way data is stored, managed, and accessed. Organizations and individuals increasingly rely on cloud platforms to handle large volumes of sensitive information, from financial records to personal data. However, as data migrates to the cloud, it faces new security threats, including unauthorized access, data breaches, and manipulation. Traditional encryption methods such as AES and RSA, though effective in securing data, rely on centralized key management systems that pose significant risks. The Advanced Encryption Standard (AES-256) is utilized for strong encryption of data files, ensuring confidentiality and protection from unauthorized access. Blockchain technology plays a pivotal role by recording file hash codes, metadata, and access logs in an immutable ledger, enabling

transparent verification and eliminating the reliance on third-party key management services. This multi-layered architecture not only protects data against brute-force, insider, and quantum-based attacks but also enhances user trust by leveraging decentralized verification mechanisms.

1.1 SCOPE OF THIS PROJECT

This project aims to:

- The scope of this project extends to developing a fully decentralized and secure cloud data protection framework that integrates blockchain technology, AES-256 encryption, audio-based steganography, and Elliptic Curve Cryptography (ECC).
- The use of audio-based Least Significant Bit (LSB) steganography enables the concealment of encryption keys within audio files, making them difficult to detect or extract.
- Authorized users will retrieve encrypted files through verified blockchain credentials and hash-based validation, ensuring that only legitimate users can access confidential data.

The Overall, this project provides a foundation for building next-generation cloud security solutions that emphasize decentralization, transparency, and resilience against advanced cyber threats, making it applicable for both personal and enterprise-level data protection scenarios.

2. PROPOSED WORK

The proposed system introduces a decentralized, tamper-proof, and multi-layered security framework to protect data stored in the cloud. It combines blockchain technology, AES-256 encryption, and audio-based steganography to overcome the vulnerabilities of traditional centralized systems.

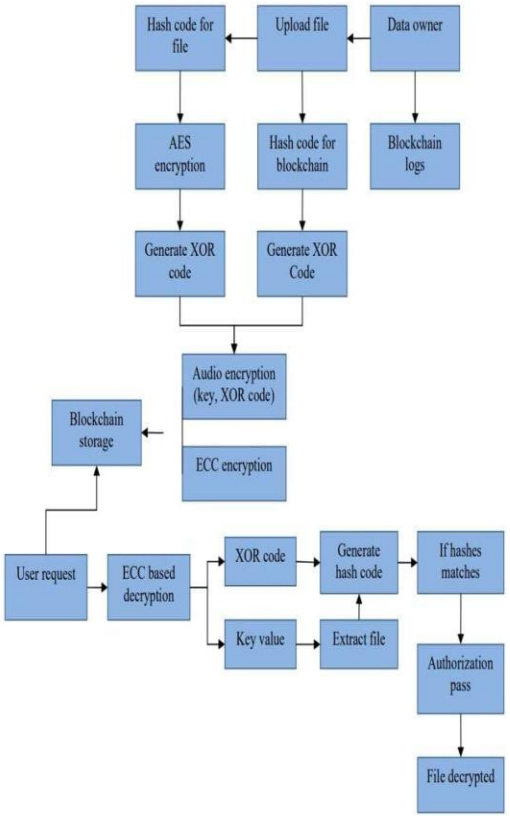
The process begins with encrypting data files using AES-256 to ensure confidentiality, followed by generating a unique hash code for each file to maintain integrity.

The system employs Elliptic Curve Cryptography (ECC) to encrypt the stego-audio files before uploading them to the cloud.

During data access, authorized users are authenticated using blockchain-verified identifiers such as Block ID, Owner ID, and Audio ID. The system decrypts the ECC-protected stego-audio file to retrieve the hidden AES key, which is then used to decrypt the original data file. Integrity verification is performed by matching the regenerated hash with the blockchain-stored hash value.

This layered security architecture eliminates centralized key management vulnerabilities while providing robust protection against insider threats, data interception, and unauthorized modifications.

2.1 System Architecture



3.1 Data Processing & Preprocessing

To ensure secure data handling, integrity verification, and reliable encryption workflows, the Cryptoacoustix framework incorporates a systematic data processing and preprocessing mechanism before cloud storage and user access.

- **File Validation:** Check file type (text / document / media) file size Remove corrupted or unsupported files.
- **Audio File Preparation:** Select carrier audio file Convert audio to standard format (e.g., WAV)Normalize audio to avoid distortion after LSB embedding.
- **Hash Preprocessing:** Compute initial SHA-256 hash of the original file Store hash temporarily before blockchain entry.
- **Audio Steganography Processing:** AES key + XOR code embedded into audio using LSB Produces stego-audio.
- **ECC Encryption:** Stego-audio encrypted using Elliptic Curve Cryptography Prevents key extraction even if audio is intercepted.

3.2 AI Secure Encryption and Blockchain-Based Protection Mechanisms

3.2.1 Hash Code Generation:

The AI Generate unique hash code (digital signature) using SHA-256 Store hash on blockchain with metadata (file ID, timestamp, owner credentials)Recalculate hash during file retrieval/decryption and compare with blockchain-stored hash.

3.2.2 Blockchain Logs:

The Generate blockchain entry for file upload, encryption, or access, store file hash code, metadata, ownership details, and access history Use distributed ledger to prevent single-entity alterations or deletions

3.3 TOOLS AND LIBRARIES

The Cryptoacoustix framework employs the following technologies:

- **Blockchain:** Ethereum, Hyperledger Fabric, or Corda for blockchain implementation.
- **Audio Encryption:** Libraries like OpenSSL, Crypto++, or NaCI for encryption algorithms
- **Cloud Systems:** AWS, Azure, or Google Cloud for cloud infrastructure.
- **Cryptographic Algorithms and Libraries:** AES-256 and Elliptic Curve Cryptography (ECC) are implemented using Python cryptographic libraries such as PyCryptodome and Cryptography, providing strong data and key encryption.
- **Python:** It is used for implementing encryption, steganography, and backend logic, while Solidity is used for writing smart contracts for blockchain operations.

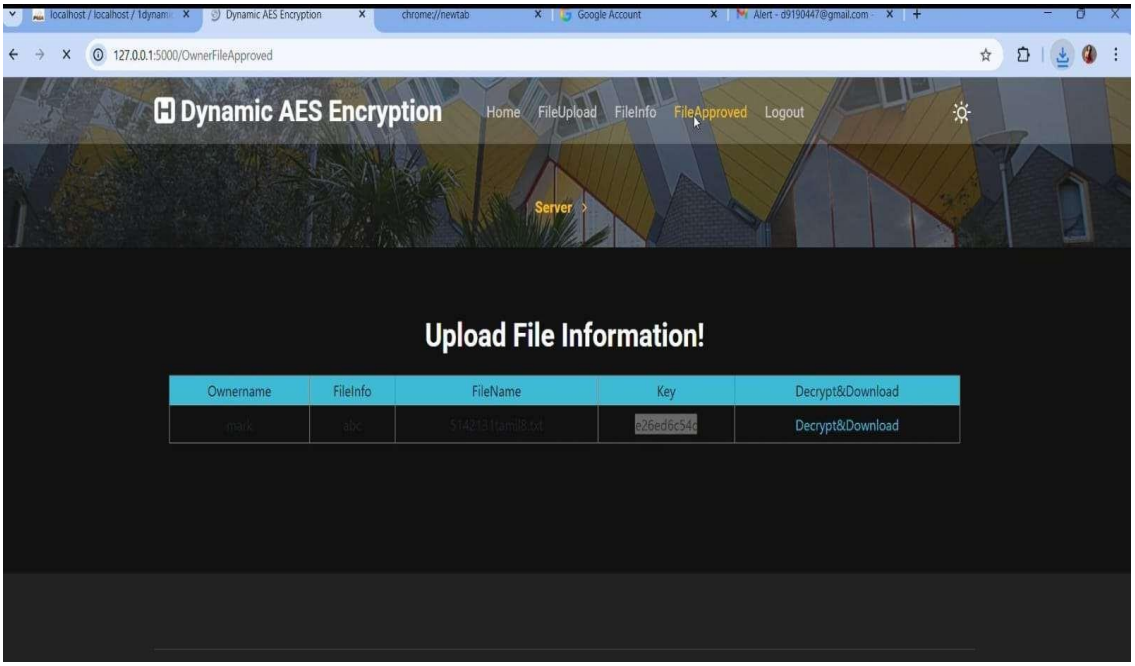
4. PROGRAM

```
from flask import Flask, render_template, flash, request, session, send_file
import mysql.connector
from ecies.utils import generate_key
from ecies import encrypt, decrypt
import base64, os
app = Flask(__name__)
app.config.from_object(__name__)
app.config['SECRET_KEY'] = '7d441f27d441f27567d441f2b6176a'
@app.route("/")
def homepage():
    return render_template('index.html')
@app.route("/ServerLogin")
def ServerLogin():
    return render_template('ServerLogin.html')
@app.route("/OwnerLogin")
def OwnerLogin():
    return render_template('OwnerLogin.html')
@app.route("/NewOwner")
def NewOwner():
    return render_template('NewOwner.html')
@app.route("/UserLogin")
def UserLogin():
    return render_template('UserLogin.html')
@app.route("/NewUser")
def NewUser():
    return render_template('NewUser.html')
@app.route("/serverlogin", methods=['GET', 'POST'])
def serverlogin():
    if request.method == 'POST':
        if request.form['uname'] == 'server' and request.form['password'] == 'server':
            conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
            cur = conn.cursor()
```

```
            cur.execute("SELECT * FROM ownertb where status='waiting'")
            data = cur.fetchall()
            conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
            cur = conn.cursor()
            cur.execute("SELECT * FROM ownertb where status!='waiting'")
            data1 = cur.fetchall()
            return render_template('ServerHome.html', data=data, data1=data1)
        else:
            flash('Username or Password is Incorrect !')
            return render_template('ServerLogin.html')
@app.route("/ServerHome")
def ServerHome():
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM ownertb where status='waiting'")
    data = cur.fetchall()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM ownertb where status!='waiting'")
    data1 = cur.fetchall()
    return render_template('ServerHome.html', data=data, data1=data1)
import hmac
import hashlib
import binascii
def  create_sha256_signature(key,  message):
    byte_key = binascii.unhexlify(key)
    message = message.encode()
    return hmac.new(byte_key, message, hashlib.sha256).hexdigest().upper()
@app.route("/Approved")
def Approved():
    id = request.args.get('lid')
    email = request.args.get('email')
    import random
    loginkey = random.randint(1111, 9999)
    message = "Owner Login Key :" + str(loginkey)
    sendmail(email, message)
    lhash  =  create_sha256_signature("E49756B4C8FAB4E48222A3E7F3B97CC3",  str(loginkey))
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cursor = conn.cursor()
    cursor.execute("Update ownertb set Status='Active',LoginKey='" + str(loginkey) + "',Hash='" + str(
        lhash) + "' where id='" + id + "' ")
    conn.commit()
    conn.close()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM ownertb where status='waiting'")
    data = cur.fetchall()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM ownertb where status='Active'")
    data1 = cur.fetchall()
    return render_template('ServerHome.html', data=data, data1=data1)
@app.route("/Reject")
def Reject():
    id = request.args.get('lid')
```

```
    email = request.args.get('email')
    message = "Your Request  Rejected"
    sendmail(email, message)
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cursor = conn.cursor()
    cursor.execute("Update ownertb set Status='reject' where id='" + id + "' ")
    conn.commit()
    conn.close()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM ownertb where status='waiting'")
    data = cur.fetchall()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM ownertb where status !='waiting'")
    data1 = cur.fetchall()
    return render_template('ServerHome.html', data=data, data1=data1)
@app.route("/SUserInfo")
def SUserInfo():
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb where status='waiting'")
    data = cur.fetchall()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb where status!='waiting'")
    data1 = cur.fetchall()
    return render_template('SUserInfo.html', data=data, data1=data1)
@app.route("/UApproved")
def UApproved():
    id = request.args.get('lid')
    email = request.args.get('email')
    import random
    loginkey = random.randint(1111, 9999)
    message = "User Login Key :" + str(loginkey)
    sendmail(email, message)
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cursor = conn.cursor()
    cursor.execute("Update regtb set Status='Active',LoginKey='" + str(loginkey) + "' where id='" + id + "' ")
    conn.commit()
    conn.close()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb where status='waiting'")
    data = cur.fetchall()
    conn = mysql.connector.connect(user='root', password='', host='localhost', database='1dynamicaesdb')
    cur = conn.cursor()
    cur.execute("SELECT * FROM regtb where status='Active'")
    data1 = cur.fetchall()
    return  render_template('SUserInfo.html',  data=data,  data1=data1)
```

5. RESULTS AND OUTPUT



- **Secure File Management Dashboard:** Displays a web-based interface for a system titled Dynamic AES Encryption, indicating the use of Advanced Encryption Standard (AES) for protecting uploaded files.
- **Navigation and Access Control:** The top navigation bar includes options such as Home, FileUpload, FileInfo, FileApproved, and Logout, suggesting role-based access and controlled file workflow.
- **Uploaded File Information Table:** Shows a structured table containing:
  - **Owner Name:** Identifies the file uploader
  - **File Info:** Descriptive metadata about the file
  - **File Name:** Name of the uploaded encrypted file
  - **Encryption Key:** A unique AES key associated with the file
  - **Decrypt & Download:** Action link to securely decrypt and retrieve the file
- **Encryption Key Visibility:** The presence of an AES key column indicates dynamic key generation per file, emphasizing confidentiality and controlled access.
- **Secure Decryption Mechanism:** The Decrypt & Download option implies that only authorized users can decrypt files, reducing the risk of unauthorized data exposure.
- **Server-Side Processing Indicator:**
  The "Server" label suggests backend validation and cryptographic operations are handled securely on the server.

6. CONCLUSION

In conclusion, the proposed decentralized cloud data security framework successfully addresses the major vulnerabilities of traditional encryption systems by integrating blockchain technology, AES-256 encryption, audio-based steganography, and Elliptic Curve Cryptography (ECC). This multi-layered approach ensures data confidentiality, integrity, and authenticity while eliminating the risks associated with centralized key management.

REFERENCES

[1] Narayanan, Uma, Varghese Paul, and Shelbi Joseph. "A novel system architecture for secure authentication and data sharing in cloud enabled Big Data Environment." Journal of King Saud University-Computer and Information Sciences 34.6 (2022): 3121-3135.

[2] Gupta, Ishu, et al. "Secure data storage and sharing techniques for data protection in cloud environments: A systematic review, analysis, and future directions." IEEE Access 10 (2022): 71247-71277.

[3] Athanere, Smita, and Ramesh Thakur. "Blockchain based hierarchical semi-decentralized approach using IPFS for secure and efficient data sharing." Journal of King Saud University-Computer and Information Sciences 34.4 (2022): 1523-1534.

[4] Prajapati, Priteshkumar, and Parth Shah. "A review on secure data deduplication: Cloud storage security issue." Journal of King Saud University-Computer and Information Sciences 34.7 (2022): 3996-4007.

[5] Ullah, Zia, et al. "Towards blockchain-based secure storage and trusted data sharing scheme for IoT environment." IEEE access 10 (2022): 36978-36994.

[6] Wang, Limei, and Yun Wang. "Supply chain financial service management system based on block chain IoT data sharing and edge computing." Alexandria engineering journal 61.1 (2022): 147-158.

[7] Zheng, Kangning, et al. "Blockchain technology for enterprise credit information sharing in supply chain finance." Journal of Innovation & Knowledge 7.4 (2022): 100256.

[8] Zhang, Rui, and Ling Liu. “Security models and requirements for healthcare application clouds.” IEEE 3rd International Conference on Cloud Computing, 2010, pp. 268–275.

[9] Li, Xiaoqiang, et al. “A blockchain-based data security framework for decentralized cloud storage.” IEEE Access 8 (2020): 205944–205955.