**Project Report: Decentralized Biometric Authentication System Using Hyperledger Fabric, IPFS, and Dlib**

**🔐 Abstract**

This project presents a decentralized biometric authentication system that integrates blockchain technology, distributed file systems, and machine learning-based facial recognition to enhance security, privacy, and tamper-proof identity management. Traditional biometric systems are centralized and vulnerable to data breaches and single points of failure. Our solution overcomes these limitations by leveraging:

* **Dlib’s Face Recognition** for feature extraction
* **IPFS** (InterPlanetary File System) for distributed storage of biometric embeddings
* **Hyperledger Fabric** as the permissioned blockchain for access control and verification
* **Smart Contracts (Chaincode)** for managing identity lifecycle events such as registration, authentication, and deletion.

**📘 1. Introduction**

Biometric systems are increasingly used for authentication due to their uniqueness and non-transferability. However, centralized biometric databases pose serious risks of mass breaches, unauthorized access, and lack of transparency. This project proposes a **Decentralized Biometric Authentication System** that addresses these issues by combining **facial feature embeddings**, **blockchain-based access control**, and **decentralized storage**.

**🧭 2. System Overview**

The system employs **dlib** for facial feature extraction, **IPFS** for decentralized embedding storage, and **Hyperledger Fabric** for secure, permissioned control over identity records. All operations—registration, confirmation, retrieval, and deletion—are governed through smart contracts.

**⚙ 3. Components**

| **Component** | **Description** |
| --- | --- |
| **Dlib** | Extracts 128D facial embeddings using a ResNet model. |
| **IPFS** | Stores face vectors as JSON using content-addressed hashes (CIDs). |
| **Hyperledger Fabric** | Acts as the blockchain ledger that records hashes and CIDs. |
| **Chaincode** | Smart contract managing lifecycle events (register, authenticate, delete). |
| **Listener Scripts** | Off-chain services that react to blockchain events to perform IPFS tasks. |
| **SVM Classifier** | Evaluates face vectors using downloaded embeddings from IPFS. |

**🔄 4. Workflow**

**📝 A. Registration**

1. Capture face image using camera or OpenCV.
2. Extract facial landmarks and compute 128D embedding via dlib\_face\_recognition\_resnet\_model\_v1.dat.
3. Compare with existing vectors using **cosine similarity threshold (0.95)**.
4. Emit **RegisterFace** event via chaincode.

bash

python3 register\_R.py

1. Unconfirmed vectors are temporarily stored in unconfirmed\_registers.json.
2. Off-chain listener script uploads vector JSON to IPFS and confirms CID on blockchain:

python3 listener.py

**🔐 B. Authentication**

1. Extract vector from a query image.
2. Download vectors from IPFS using CID fetched from blockchain.
3. Compute cosine similarity with each stored vector.
4. Authenticate if threshold is satisfied.

**❌ C. Deletion**

1. Detect face and emit **RequestDeleteCIDRecord** event:

bash

python3 delete.py

1. Listener script checks for CID match and deletes from both IPFS and blockchain:

bash

python3 listener\_del.py

**🧱 5. Data Flow Architecture**

+------------------+

| Face Image |

+--------+---------+

|

Dlib Embedding

|

v

+-----------------------------+

| SHA-256 Hash Computation |

+-----------------------------+

|

+-------------------+--------------------+

| |

v v

Blockchain <--- ConfirmCIDUpload ---- IPFS

(Stores hash & CID) (Stores vector JSON)

^

Listener monitors chaincode events

**🧠 6. Smart Contract Logic**

**Chaincode Functions Implemented in cidrecord.go:**

go

RegisterFace(id string, vector\_json string) // Emits event

ConfirmCIDUpload(id string, cid string) // Confirms IPFS upload

RequestDeleteCIDRecord(id string, cid string) // Emits deletion event

DeleteCIDRecord(id string) // Deletes record

ReadCIDRecord(id string) (\*CIDRecord, error) // Returns single record

GetAllCIDRecords() ([]\*CIDRecord, error) // Lists all records

🔒 **Security**: All IPFS operations are mediated through smart contracts.

🛂 **Access Control**: Only permitted MSPs (e.g., Org1MSP) can invoke chaincode.

**⛓ 7. Blockchain Setup**

Hyperledger Fabric is deployed using:

bash

cd fabric-samples/test-network

./network.sh up createChannel -ca

./network.sh deployCC -ccn cidrecord -ccp ../chaincode/cidrecord -ccl go

Querying CID records from blockchain:

bash

peer chaincode query -C mychannel -n cidrecord -c '{"function":"ReadCIDRecord","Args":[""]}'

**🛰 8. IPFS Integration**

Upload vector to IPFS:

bash

ipfs add -Q vector.json

Retrieve from IPFS:

bash

ipfs cat <CID>

Remove CID from local IPFS node:

bash

ipfs pin rm <CID>

**📈 9. Biometric Evaluation Metrics**

Evaluation is done using the **LFW dataset** and SVM classifiers trained on IPFS-downloaded vectors.

| **Metric** | **Description** |
| --- | --- |
| **Accuracy** | Correct predictions / Total predictions |
| **FAR** | False Acceptance Rate |
| **FRR** | False Rejection Rate |
| **EER** | Equal Error Rate (FAR = FRR) |
| **ROC-AUC** | Area under ROC curve |
| **Confusion Matrix** | TP, FP, TN, FN distribution |

**📂 10. Files in System**

| **File Name** | **Purpose** |
| --- | --- |
| register\_R.py | Registers new faces and checks for duplicates |
| listener.py | Uploads vectors to IPFS and confirms CIDs |
| listener\_del.py | Deletes confirmed vectors from IPFS and blockchain |
| delete.py | Emits deletion event on face match |
| cidrecord.go | Chaincode for managing vector registration lifecycle |

**✅ 11. Features & Guarantees**

* ❌ No local vector storage (in-memory only)
* ✅ All uploads confirmed via smart contract
* ✅ CID deduplication using cosine similarity
* ✅ Chaincode-only access to register, read, or delete data
* ✅ Modular event-driven listener architecture

**🔒 12. Security Considerations**

| **Feature** | **How It’s Addressed** |
| --- | --- |
| **Privacy** | Only embeddings (not raw images) stored |
| **Immutability** | Blockchain records hash → CID permanently |
| **Tamper Resistance** | Any hash mismatch invalidates the record |
| **Access Control** | Only permitted MSPs can modify records |
| **No Central Storage** | IPFS ensures decentralized persistence |

**📈 13. Optimization Techniques**

* ✅ **FAISS** used for fast approximate nearest neighbor search.
* ✅ **Blockchain-mediated vector upload** (no direct IPFS access)
* ✅ **Event-driven off-chain listeners**
* ✅ **Polling fallback** for environments without CA/EventHub

**⚠ 14. Limitations & Future Work**

| **Limitation** | **Potential Fix** |
| --- | --- |
| Few samples for some faces | Introduce minimum registration threshold |
| No encryption for embeddings | Use AES/RSA to encrypt vector before IPFS upload |
| IPFS performance bottlenecks | Use IPFS cluster or pinning services |
| Vectors lack label for training | Add label field in JSON during registration |

**✅ 15. Conclusion**

This project presents a secure, scalable, and tamper-proof decentralized biometric authentication system that:

* Extracts facial features using **dlib**
* Stores biometric vectors on **IPFS**
* Maintains audit trails and control on **Hyperledger Fabric**
* Performs evaluations using **real datasets** like LFW

This approach supports **privacy-preserving KYC**, **access control**, and can be extended to **multimodal biometric systems** (e.g., iris, fingerprint) with minimal architectural change.

Result for LFW dataset:

iometric@biometric:~/1$ source venv/bin/activate

(venv) biometric@biometric:~/1$ python3 acc1.py

📥 Downloading vectors from IPFS via blockchain records...

100%|███████████████████████████████████████| 3600/3600 [01:17<00:00, 46.22it/s]

🔢 Loaded 450 vectors

🔍 Computing pairwise similarity...

100%|████████████████████████████████████████| 450/450 [00:00<00:00, 565.79it/s]

🎯 Evaluation Metrics:

Accuracy: 98.31%

AUC: 0.9957

EER: 0.0169 (at threshold 0.9181)

FAR: 0.0169

FRR: 0.0170

Confusion Matrix:

[[95902 1644]

[ 59 3420]]

📊 Graphs saved in ./evaluation\_plots/

**Accuracy graph:**

A graph of a normalized curve

AI-generated content may be incorrect.

**Heat Map for confusion matrix:A blue and white graph

AI-generated content may be incorrect.**

**AUC-ROC Curve:**

**A graph of a curve

AI-generated content may be incorrect.**