Blockchain-Based Credible and Secure Education Data Management System

A comprehensive system for managing academic records using blockchain technology, ensuring data integrity, authenticity, and tamper-proof verification.

Features

- **Tamper-Proof Storage**: Academic records anchored on Hyperledger Fabric blockchain
- **Data Privacy**: Symmetric encryption (Fernet) for sensitive information
- **Batch Processing**: Merkle tree implementation for efficient batch record anchoring
- **Instant Verification**: Fast credential verification using cryptographic hashes
- **Role-Based Access**: Secure access control for institutions, students, and verifiers
- **Audit Trail**: Complete transaction history with immutable logging

Architecture

The system uses a three-tier architecture:

1. **Presentation Layer**: RESTful API (Flask)
2. **Application Layer**: Business logic, encryption, and blockchain interaction
3. **Data Layer**: Off-chain encrypted database (SQLite) + Hyperledger Fabric

Prerequisites

- Python 3.10+
- Docker and Docker Compose
- Hyperledger Fabric 2.x
- Go 1.20+ (for chaincode)

Installation

1. Clone the Repository

bash

```
git clone <repository-url>
cd blockchain-education-system
```

2. Set Up Python Environment

bash

```
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

3. Configure Environment Variables

Create a `.env` file:

env

```
SECRET_KEY=your-secret-key-here
FERNET_KEY=your-fernet-key-here
DEBUG=True
DATABASE_URL=sqlite:///education_records.db
INSTITUTION_ID=inst123
INSTITUTION_NAME=Cambridge Institute of Technology
```

Generate a Fernet key:

python

```
from cryptography.fernet import Fernet
print(Fernet.generate_key())
```

4. Initialize Database

bash

```
cd app
python -c "from models import init_db; init_db()"
```

5. Set Up Hyperledger Fabric (Optional for Mock Mode)

For development, the system runs in mock mode. For production:

bash

```
# Follow Hyperledger Fabric installation guide
# Deploy the chaincode from chaincode/education_contract.go
```

Running the Application

Development Mode (Mock Blockchain)

bash

```
cd app
python app.py
```

The API will be available at `http://localhost:5000`

Production Mode (With Fabric)

1. Start Hyperledger Fabric network
2. Deploy chaincode
3. Set `mock_mode = False` in `blockchain_adapter.py`
4. Start the Flask application

API Endpoints

Student Management

Create Student

http

```
POST /api/students
Content-Type: application/json

{
  "student_id": "S12345",
  "name": "John Doe",
  "email": "john@example.com",
  "public_metadata": {
    "program": "Computer Science",
    "year": 2024
  }
}
```

Get Student

http

```
GET /api/students/{student_id}
```

Record Management

Issue Record

http

```
POST /api/issue
Content-Type: application/json

{
 "student_id": "S12345",
 "payload": {
  "course": "CS301",
  "grade": "A",
  "semester": "2024-2",
  "credits": 4
 }
}
```

Response:

json

```
{
 "status": "issued",
 "record_id": "rec_abc123...",
 "tx_id": "tx_xyz789...",
 "payload_hash": "2c4f3d9a..."
}
```

Get Record (Encrypted)

http

```
GET /api/records/{record_id}
```

Decrypt Record (Authorized)

http

```
GET /api/records/{record_id}/decrypt
```

Verify Record

http

```
GET /api/verify/{record_id}
```

Response:

json


```
{
 "verified": true,
 "record_id": "rec_abc123",
 "payload_hash": "2c4f3d9a...",
 "recomputed_hash": "2c4f3d9a...",
 "onchain_anchor": {
  "recordID": "rec_abc123",
  "anchor": "2c4f3d9a...",
  "issuer": "inst123",
  "time": "2025-10-30T10:30:00Z"
 },
 "tx_id": "tx_xyz789"
}
```


Batch Issue

http

```
POST /api/batch-issue
Content-Type: application/json

{
 "records": [
  {
   "student_id": "S12345",
   "payload": {"course": "CS301", "grade": "A"}
  },
  {
   "student_id": "S12346",
   "payload": {"course": "CS302", "grade": "B+"}
  }
 ]
}
```

Transactions

Get All Transactions


http

```
GET /api/transactions
```

Health Check


http

```
GET /api/health
```

Usage Examples

Python Example


python

```python
import requests

# Create a student
response = requests.post('http://localhost:5000/api/students', json={
    'student_id': 'S12345',
    'name': 'John Doe',
    'email': 'john@example.com'
})

# Issue a record
response = requests.post('http://localhost:5000/api/issue', json={
    'student_id': 'S12345',
    'payload': {
        'course': 'CS301',
        'grade': 'A',
        'semester': '2024-2'
    }
})

record_id = response.json()['record_id']

# Verify the record
response = requests.get(f'http://localhost:5000/api/verify/{record_id}')
print(response.json())
```

cURL Example

bash

```bash
# Create student
curl -X POST http://localhost:5000/api/students \
  -H "Content-Type: application/json" \
  -d '{
    "student_id": "S12345",
    "name": "John Doe",
    "email": "john@example.com"
  }'

# Issue record
curl -X POST http://localhost:5000/api/issue \
  -H "Content-Type: application/json" \
  -d '{
    "student_id": "S12345",
    "payload": {
      "course": "CS301",
      "grade": "A"
    }
  }'
```

Testing

bash

```bash
# Run unit tests
python -m pytest tests/

# Test API endpoints
python -m pytest tests/test_api.py
```

Security Considerations

Production Deployment

1. **Key Management**:
   - Use hardware security modules (HSM) or key vaults
   - Rotate encryption keys regularly
   - Never commit keys to version control
2. **Authentication**:
   - Implement OAuth2/SAML for institutional SSO
   - Add JWT-based API authentication
   - Use role-based access control (RBAC)
3. **Network Security**:
   - Enable mutual TLS for Fabric communication

    - Use HTTPS for all API endpoints
    - Implement rate limiting
4. **Data Protection**:
    - Follow GDPR/FERPA compliance
    - Implement data retention policies
    - Regular security audits

Performance Metrics

Based on prototype evaluation:

- **Transaction Latency**: 1.21s - 15.47s (batch sizes 10-500)
- **Throughput**: 8-32 tx/s
- **Storage Overhead**: ~20% (encrypted vs plaintext)
- **Verification Success Rate**: 100%

Project Structure

```
blockchain-education-system/
├── app/
│   ├── __init__.py
│   ├── app.py              # Flask application and routes
│   ├── models.py           # Database models
│   ├── crypto.py           # Encryption and hashing
│   ├── merkle.py           # Merkle tree implementation
│   ├── blockchain_adapter.py  # Blockchain interaction
│   └── config.py           # Configuration
├── chaincode/
│   └── education_contract.go  # Hyperledger Fabric chaincode
├── tests/
│   ├── test_crypto.py
│   ├── test_merkle.py
│   └── test_api.py
├── requirements.txt
├── docker-compose.yml
└── README.md
```

Future Enhancements

- [ ] W3C Verifiable Credentials integration
- [ ] Decentralized Identity (DID) support
- [ ] Zero-Knowledge Proofs for privacy
- [ ] Mobile application
- [ ] AI-driven fraud detection
- [ ] Multi-institution consortium support

Contributing

1. Fork the repository
2. Create a feature branch
3. Commit your changes
4. Push to the branch
5. Create a Pull Request

License

Authors

- Raghu P
- Shashank S
- Vittal R Babu
- Gagan R
- Supreeth V
- Sathish V

Supervisor: Prof. Aparna N

Acknowledgments

Special thanks to the Department of Information Science and Engineering, Cambridge Institute of Technology, Bengaluru, for their support and guidance throughout this project.

References

1. Hyperledger Fabric Documentation: https://hyperledger-fabric.readthedocs.io
2. Cryptography.io: https://cryptography.io
3. W3C Verifiable Credentials: https://www.w3.org/TR/vc-data-model/

Support

For issues and questions, please contact:

- Email: raghu.ise@cambridge.edu.in
- Repository Issues: [GitHub Issues]

Note: This system is currently in prototype stage. For production deployment, ensure proper security audits, key management, and compliance with relevant data protection regulations.