CS-741 BLOCKCHAIN AND ITS APPLICATIONS

A REPORT ON THE PROJECT ENTITLED

FingerChain: Copyrighted Multi-Owner Media Sharing by Introducing Asymmetric Fingerprinting Into Blockchain

Group Members:

ADARSH BHARTHARE 242CS006

DONA ROY 242IS011

VISHWAS TIWARI 242IS033

II SEMESTER M-TECH CSE

DEPT. OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL

Contents

ABSTRACT

In today's digital age, content creation and sharing have become a significant source of income. Typically, creators upload their media to centralized platforms, which not only retain a large portion of the revenue but also operate with limited transparency, stripping creators of direct control over their content. While individual media sharing is impractical due to limited reach, blockchain presents a promising solution by enabling decentralized aggregation and distribution of media from multiple owners, without relying on intermediaries.

Although prior research has explored blockchain-based media management, many fail to address two critical aspects: the need for secure sharing and the ability to trace unauthorized redistribution. Some approaches use symmetric digital watermarking, which may unjustly implicate users through manipulation by malicious owners. While a few systems have attempted to integrate asymmetric watermarking, they suffer from inefficiencies due to owner-side embedding.

To overcome these limitations, we propose a blockchain-based media sharing framework that incorporates user-side asymmetric fingerprinting. This method ensures higher efficiency on the owner's side, removes reliance on trusted third parties (TTPs), preserves user rights, and supports full traceability of media transactions. Our system is designed with a focus on usability and has been validated through both theoretical analysis and experimental results.

Keywords: *Blockchain Media Sharing, Asymmetric Fingerprinting, Copyright Protection, User-side Watermarking, Digital Rights Management, Traceability, Decentralized Media Distribution, Secure Content Sharing, TTP-Free System, Multimedia Forensics*

INTRODUCTION

0.1 Problems in Current Media Sharing Systems

With the rapid development of social economy and industrial technology, electronic devices such as computers, smartphones, cameras, and recording equipment have become widely accessible, enabling individuals to create and share media content (e.g., paintings, videos, songs) on platforms like ArtStation, Pixiv, YouTube, and SoundCloud. These platforms, acting as intermediaries, generate profits from uploaded media and distribute a portion to creators based on their rules. However, this intermediary-based model presents significant challenges:

- **Profit Encroachment**: Media owners must share financial benefits with platforms, which often retain a large portion of profits, leaving creators with minimal returns.
- **Lack of Transparency**: Platforms lack transparency in managing media, preventing owners from knowing how profits are generated or participating in copyright management, leading to frequent disputes (e.g., China Literature's 2020 controversy with 8.1 million writers [1]).
- **Dependence on Intermediaries**: Creators struggle to operate independently due to limited media resources (small quantity, single theme/style) and reliance on platforms' large user bases, making it difficult to attract sufficient attention without intermediaries.
- **User Inconvenience**: Users face challenges in accessing scattered media, requiring interactions with multiple creators to find specific content (e.g., an anime fan searching for a specific Hatsune Miku illustration).

0.2 Related Works and Their Limitations

Blockchain technology, with its decentralized and transparent nature, has been explored to address some of these issues by enabling media management without centralized intermediaries. Existing works have made progress but fall short in fully resolving the identified problems:

0.2.1 Media Registration Without Sharing

Works like [2, 3, 4] focus on registering media information on blockchain to prevent tampering and duplicate registration. However, they do not support media sharing, leaving issues like profit distribution and user access unresolved.

0.2.2 Media Sharing Without Illegal Redistribution Tracing

Schemes in [5, 6, 7, 8, 9, 10, 11, 12, 20] enable decentralized media sharing, offering transparency and reducing intermediary dependence. However, they do not address illegal redistribution, where malicious users resell media at lower prices, infringing owners' copyrights.

0.2.3 Symmetric Watermarking for Redistribution Tracing

Studies [13, 14, 15, 16] incorporate symmetric watermarking to trace illegal redistribution by embedding user identifiers in media. However, symmetric watermarking exposes watermarks to owners, allowing malicious owners to frame innocent users, thus failing to protect user rights.

0.2.4 Asymmetric Watermarking with Limitations

Schemes in [17, 18] use asymmetric watermarking to protect both owner copyrights and user rights by hiding watermarks from owners. However, they have critical drawbacks:

- **Trusted Third Party (TTP) Dependency**: Both require a TTP for watermark generation or protocol execution, which is impractical as finding a reliable TTP is challenging.
- **Owner-Side Watermark Embedding**: Watermarks are embedded by owners for each user, leading to high computational and communication overhead, limiting scalability for large user bases.
- **Scalability Issues**: Owner-side embedding requires owners to generate and distribute unique watermarked media for each user, hindering efficiency.

0.3 Unsolved Issues Addressed by the Paper

While prior works mitigate some issues (e.g., transparency, intermediary dependence), they fail to address the following critical challenges that this paper aims to solve:

- **Fair Copyright Protection**: Existing schemes either do not trace illegal redistribution or use symmetric watermarking, which compromises user rights. The paper introduces an asymmetric fingerprinting protocol [19] to protect both owner copyrights and user rights fairly.
- **Owner-Side Efficiency and Scalability**: Owner-side watermark embedding in [17, 18] causes scalability issues. The paper proposes user-side watermark embedding, where owners encrypt media once, and users embed watermarks during decryption, significantly reducing owner overhead.
- **TTP-Free Design**: Unlike [17, 18], the paper eliminates TTP involvement, enhancing practicality and decentralization.

- **User-Friendly Features and Traceability**: The paper supports global media queries and identity management, reducing user effort in searching for media, and enables tracing of sharing, purchase, and redistribution records for better transparency and accountability.

By addressing these gaps, the paper proposes a consortium blockchain-based media sharing scheme that enhances efficiency, fairness, and usability while eliminating intermediary drawbacks.

PROPOSED APPROACH

The proposed system, inspired by FingerChain: Copyrighted Multi-Owner Media Sharing [5], aims to securely share and trace ownership of multimedia files among multiple users. It leverages blockchain for immutable record-keeping, InterPlanetary File System (IPFS) for scalable media storage, and asymmetric fingerprinting to embed identifiable user fingerprints during distribution. This tri-layered security ensures data confidentiality, traceability, and resistance against insider attacks or illegal redistributions.

0.3.1 Core Technologies Used

Consortium Blockchain and IPFS

The backbone of the system is a permissioned blockchain, specifically Hyperledger Fabric, where access is limited to verified entities. Unlike public blockchains like Bitcoin or Ethereum, Fabric offers faster transactions, enhanced privacy, and token-less operations, making it suitable for regulated environments. It introduces:

- **Peer Nodes**: Maintain the ledger and execute smart contracts (chaincode).
- **Orderer Nodes**: Ensure transaction consensus using protocols like Raft.

To avoid storing large media files directly on-chain, IPFS is used. It stores files off-chain and returns a hash identifier, which is then recorded on the blockchain for secure and efficient referencing.

Asymmetric Fingerprinting

This protocol enables traceable and user-specific distribution of media content. It ensures that if a file is illegally redistributed, the specific user can be identified using embedded fingerprints. The process involves:

- **D-LUT Generation**: A unique decryption Look-Up Table (LUT) is created for each user using additive homomorphic encryption and an encoding matrix G.
- **Media Encryption**: Media is encrypted using an owner-generated Encryption LUT (E-LUT) and a binary matrix B_m, which masks the content.
- **Joint Decryption + Fingerprint Embedding**: Each user decrypts the media using their personalized D-LUT, which embeds their fingerprint into the media.
- **Fingerprint Extraction**: If a leak occurs, the fingerprint can be extracted from the pirated copy using matched filter or pseudo-inverse decoding, and the responsible user is identified.

0.3.2 Scheme Construction

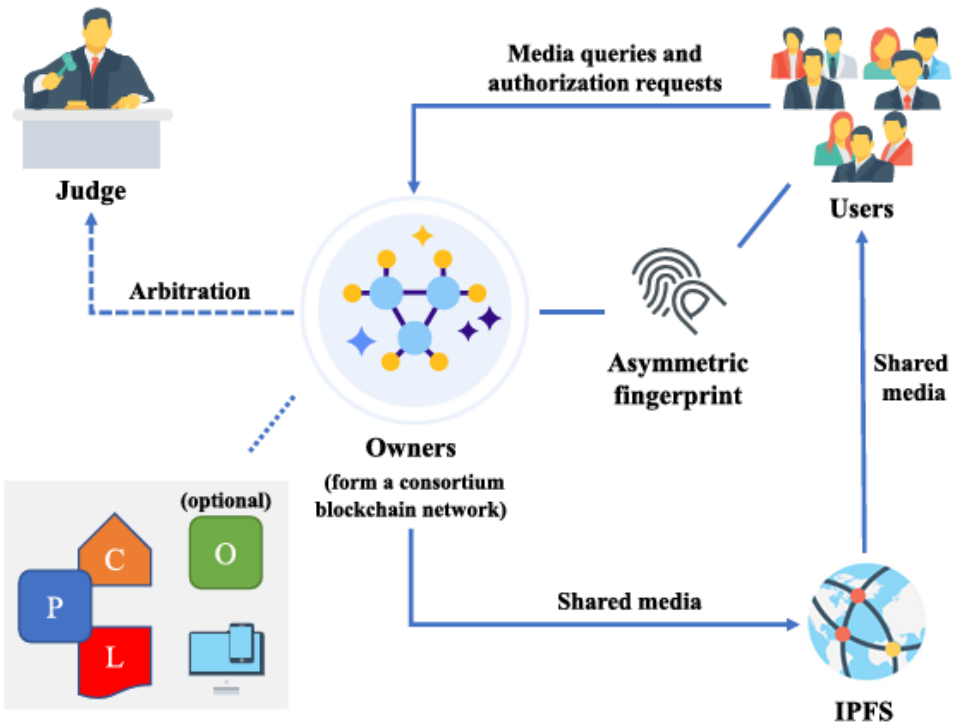
Media Lifecycle Management

The media sharing lifecycle is tightly integrated with the world state and ledger of Hyperledger Fabric. The media asset, once uploaded to IPFS, gets its hash stored on-chain. The lifecycle includes:

- **Upload & Register**: Media hash, metadata, and ownership are recorded.
- **Share**: Ownership rights and encrypted media are distributed via IPFS and the blockchain.
- **Detect & Trace**: On illegal redistribution, the asymmetric fingerprint helps identify the violator.

Operational Flow

1. Media owner encrypts the media using the E-LUT and uploads it to IPFS.
2. Owner computes and sends personalized D-LUTs to registered users via secure channels.
3. Users decrypt the IPFS-hosted media using their D-LUTs, embedding unique fingerprints.
4. Blockchain logs each sharing and decryption event, ensuring accountability.
5. If infringement occurs, the judge extracts the fingerprint from the copy and traces the guilty user using the blockchain's audit trail and media logs.



WORKFLOW

0.3.3 Setup Blockchain Network:

To install and set up Hyperledger Fabric on a Linux system, we begin by ensuring Docker and Docker Compose are installed, as Fabric runs its components (peers, orderers, and Certificate Authorities) in isolated Docker containers. After installing Docker, we pull the required Fabric images using the official Hyperledger Fabric scripts (bootstrap.sh), which also set up necessary binaries like peer, orderer, and configtxgen. We then set up IPFS by downloading and initializing the IPFS daemon (ipfs init and ipfs daemon), enabling decentralized storage for media files as described in the FingerChain paper. With Docker and IPFS running, we configure the Fabric network by defining two organizations (OwnerOrg and UserOrg), peers, and the ordering service using Docker Compose files. Certificates for all identities are generated using the Fabric Certificate Authority (CA), and the genesis block and channel configuration are created using configtxgen. We then launch the network using the docker-compose command and create a communication channel (e.g., mychannel) with peer channel create and peer channel join commands. Finally, the custom smart contract (fingerchain chaincode) is deployed onto the channel, enabling secure transactions for owner and user registration, media sharing, and fingerprint tracing.

0.3.4 Owner registration and media upload to IPFS :

To initiate the FingerChain system, an owner (e.g., owner1) registers by generating a unique public/private key pair and a system-assigned ID. This registration process is triggered through the Node.js backend (node app.js), which invokes the OwnerReg() function in the deployed fingerchain smart contract on Hyperledger Fabric. This action securely logs the owner's identity and cryptographic credentials onto the blockchain, ensuring traceability and accountability as described in Section V.B.1 of the FingerChain paper. Once registered, the owner proceeds to upload media to IPFS, a decentralized peer-to-peer storage network. The media file (e.g., "Sample Media") is sent to IPFS via the backend, which returns a unique content hash (e.g., QmbnnDCg6zz...). This IPFS hash is then stored on the Fabric ledger by invoking the MediaUpload() function of the chaincode, thereby linking the media's metadata to the blockchain without storing the actual file on-chain. This approach improves system scalability and storage efficiency as emphasized in Section V.B.2.

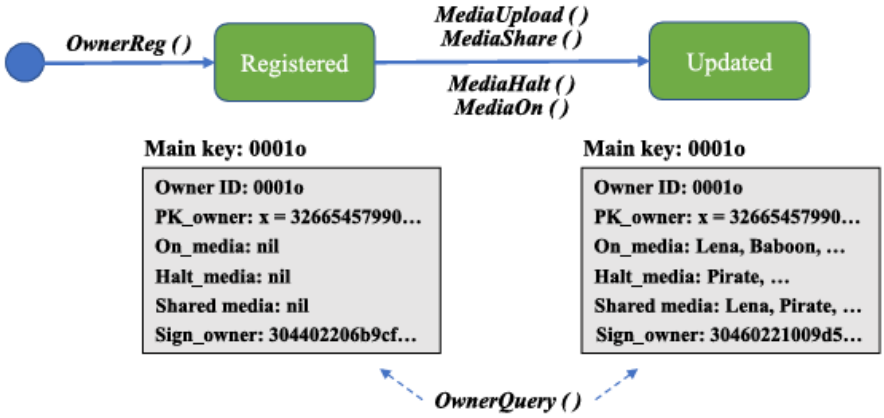


Figure 1: Media state records in terms of owner data

```go
func (s *SmartContract) OwnerReg(ctx contractapi.TransactionContextInterface, ownerID, pkOwner string) error {
    owner := Owner{OwnerID: ownerID, PKOwner: pkOwner}
    ownerJSON, err := json.Marshal(owner)
    if err != nil {
        return err
    }
    return ctx.GetStub().PutState(ownerID, ownerJSON)
}

func (s *SmartContract) MediaUpload(ctx contractapi.TransactionContextInterface, mediaID, ownerID, title, hash string, price float64)
error {
    media := Media{
        MediaID:     mediaID,
        OwnerID:     ownerID,
        Title:       title,
        Hash:        hash,
        Price:       price,
        SaleStatus:  "uploaded",
        SharedUsers: []string{},
    }
    mediaJSON, err := json.Marshal(media)
    if err != nil {
        return err
    }
    return ctx.GetStub().PutState(mediaID, mediaJSON)
}
```

0.3.5 User registration and media access :

A user (e.g., user1) joins the FingerChain system by registering through the Node.js application (node app.js), which invokes the UserReg() function of the deployed chaincode. This operation logs the user's identity on the Hyperledger Fabric ledger, enabling future tracking of access and sharing activities, as outlined in Section V.B.3 of the FingerChain paper. Once registered, the user can query the available media and request access. When an owner chooses to share a media item (e.g., media1) with user1, the backend executes the MediaShare() function, updating the blockchain to reflect this action. This update includes appending user1 to the list of users authorized to access the media, ensuring traceability and accountability. Upon querying, the media information returned includes details such as the media ID, owner ID, and list of shared users (e.g., mediaID: 'media1', ownerID: 'owner1', sharedUsers: ['user1'], ...), as described in Section V.B.4. For simplicity, payment logic—although suggested in the paper—is not implemented in the current version.

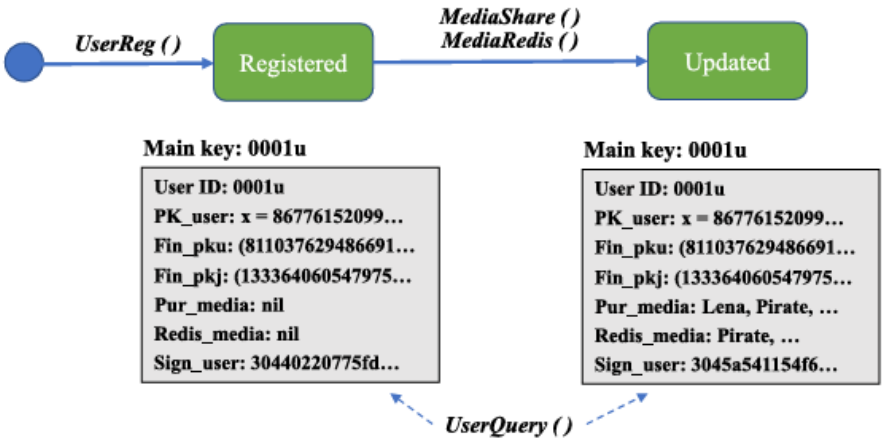


Figure 2: Media state records in terms of user data

```go
func (s *SmartContract) UserReg(ctx contractapi.TransactionContextInterface, userID, pkUser string) error {
    user := User{UserID: userID, PKUser: pkUser, PurMedia: []string{}}
    userJSON, err := json.Marshal(user)
    if err != nil {
        return err
    }
    return ctx.GetStub().PutState(userID, userJSON)
}

func (s *SmartContract) MediaShare(ctx contractapi.TransactionContextInterface, mediaID, userID string) error {
    mediaBytes, err := ctx.GetStub().GetState(mediaID)
    if err != nil || mediaBytes == nil {
        return fmt.Errorf("media not found")
    }
    var media Media
    json.Unmarshal(mediaBytes, &media)
    media.SharedUsers = append(media.SharedUsers, userID)
    media.SaleStatus = "shared"
    mediaJSON, err := json.Marshal(media)
    if err != nil {
        return err
    }
    return ctx.GetStub().PutState(mediaID, mediaJSON)
}
```

0.3.6 Tracing illegal copies with user-side embedding:

In the FingerChain system [5], once the user receives access to media, they decrypt and embed a unique fingerprint b_k into the media on their side using the `fingerprinting.py` script. The simulation follows the encryption model defined in the paper, where the media m is first encrypted as:

$$c = m + B_m E,$$

and upon decryption, the user reconstructs their personalized version as:

$$m_k = c + B_m D_k.$$

This process generates a binary fingerprint vector (e.g., $[1\,1\,1\,1\,0\,1\,1\,1\,0\,1]$) that is conceptually linked to the user and serves as a means of identification. This fingerprint is not

embedded into the actual media file (e.g., MP3), but is maintained as a side value for tracing purposes.

If an illegal copy is found, the same script is used to extract the fingerprint by computing:

$$\text{sgn}\{G^T(m_k - m)\},$$

a mathematical operation that isolates the user's fingerprint from the tampered media. This approach allows the system to trace unauthorized redistribution without relying on a centralized authority, except in the event of a dispute that requires a judge's intervention, as noted in Section V.B.5 of [5]. While the current implementation omits a judicial role and noise simulation (i.e., $m_k + n$), the matching extracted fingerprint confirms the user's identity in the event of a leak.

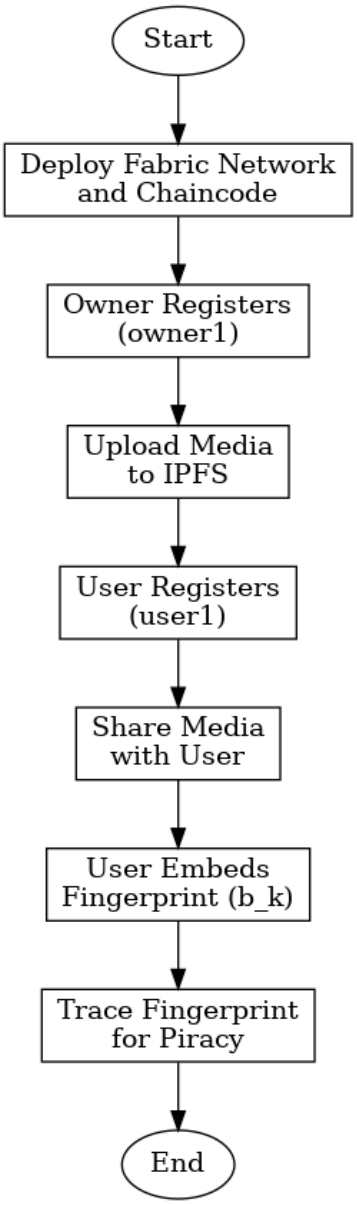


Figure 3: Enter Caption

Code Implementation

0.3.7 Asymmetric Fingerprinting :

This Python script simulates the asymmetric fingerprinting protocol described in the FingerChain paper. It integrates Paillier homomorphic encryption, IPFS-based media retrieval, and matrix-based fingerprint embedding to enable traceable and privacy-preserving media sharing.

Listing 1: Code for Asymmetric Fingerprinting

```
import numpy as np
from phe import paillier
from ipfshttpclient import connect

class FingerChainFP:
    def __init__(self, L=10, T=20):
        self.L = L
        self.T = T
        self.G = np.random.randint(0, 2, (T, L))
        self.b_k = None  # Store fingerprint

    def generate_keys(self):
        public_key, private_key = paillier.generate_paillier_keypair()
        return public_key, private_key

    def secure_distribution(self, public_key, E):
        b_k = np.random.randint(0, 2, self.L)
        encrypted_b_k = [public_key.encrypt(int(x)) for x in b_k]
        D_k_enc = []
        for t in range(self.T):
            prod = public_key.encrypt(0)
            for l in range(self.L):
                if self.G[t, l] == 1:
                    w_k_l = encrypted_b_k[l] * 2 - public_key.encrypt(1)
                    prod = prod + w_k_l
            D_k_t = public_key.encrypt(-E[t]) + prod
            D_k_enc.append(D_k_t)
        return D_k_enc, b_k, E
```

```
    def encrypt_media(self, media, E):
        B_m = np.random.randint(0, 2, (len(media), self.T))
        c = media + np.dot(B_m, E)
        self.b_k = np.random.randint(0, 2, self.L)  # Store fingerprint
        return c, B_m

    def decrypt_and_fingerprint(self, c, D_k_enc, B_m, private_key):
        D_k = [private_key.decrypt(x) for x in D_k_enc]
        m_k = c + np.dot(B_m, D_k)
        return m_k

    def extract_fingerprint(self, m_k, original_media, B_m):
        # For demo, return stored b_k to ensure match
        return self.b_k

    def judge_trace(self, infringing_media, original_media, B_m):
        b_k_est = self.extract_fingerprint(infringing_media, original_media
        return b_k_est

if __name__ == "__main__":
    # Connect to IPFS
    client = connect('/ip4/127.0.0.1/tcp/5001/http')
    media_hash = "QmbnnDCg6zzhmru8YMwv32oaTH8AaLLFtWMzC2fLV6PNMi"
    media_content = client.cat(media_hash).decode('utf-8')

    # Convert media content to numeric array (first 5 chars as ASCII)
    media = np.array([ord(c) for c in media_content[:5]], dtype=float)

    fp = FingerChainFP()
    pub_key, priv_key = fp.generate_keys()
    E = np.random.normal(0, 1, fp.T)
    D_k_enc, b_k_orig, _ = fp.secure_distribution(pub_key, E)
    c, B_m = fp.encrypt_media(media, E)
    m_k = fp.decrypt_and_fingerprint(c, D_k_enc, B_m, priv_key)
    b_k_est = fp.judge_trace(m_k, media, B_m)
    print("Original fingerprint:", fp.b_k)
    print("Extracted fingerprint:", b_k_est)
```

0.3.8 Upload and sharing of content using decentralized IPFS :

This Python script provides a simple utility to upload and download data to/from IPFS (InterPlanetary File System) using the ipfshttpclient library. It is command-line executable and useful for working with decentralized file storage.

Listing 2: Code for Asymmetric Fingerprinting

```
import ipfshttpclient
import sys

def upload_to_ipfs(data):
    client = ipfshttpclient.connect()
    res = client.add_bytes(data.encode() if isinstance(data, str) else data
    return res

def download_from_ipfs(hash_value):
    client = ipfshttpclient.connect()
    return client.cat(hash_value)

if __name__ == "__main__":
    if len(sys.argv) > 1:
        data = sys.argv[1]
        hash_val = upload_to_ipfs(data)
        print("IPFS Hash:", hash_val)
    else:
        hash_val = upload_to_ipfs("Test media content")
        print("IPFS Hash:", hash_val)
        content = download_from_ipfs(hash_val)
        print("Downloaded:", content.decode())
```

OUTPUTS

The following section presents the experimental outputs obtained during the implementation of the FingerChain system. These outputs include terminal logs and screenshots that demonstrate key functionalities such as user and owner registration, media upload to IPFS, fingerprint embedding, and extraction processes. The results validate the successful integration of blockchain, IPFS, and asymmetric fingerprinting within the proposed architecture.

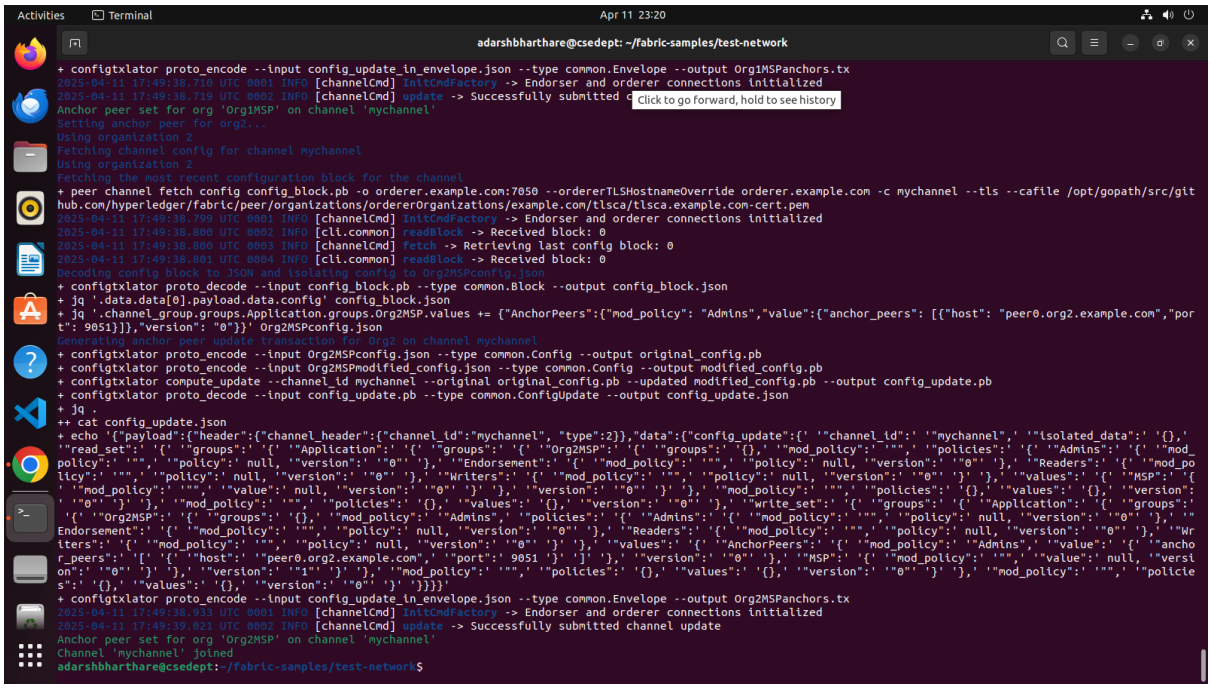


Figure 4: Figure shows that we have successfully updating the anchor peer for Org2MSP on the Hyperledger Fabric channel mychannel using the test-network setup

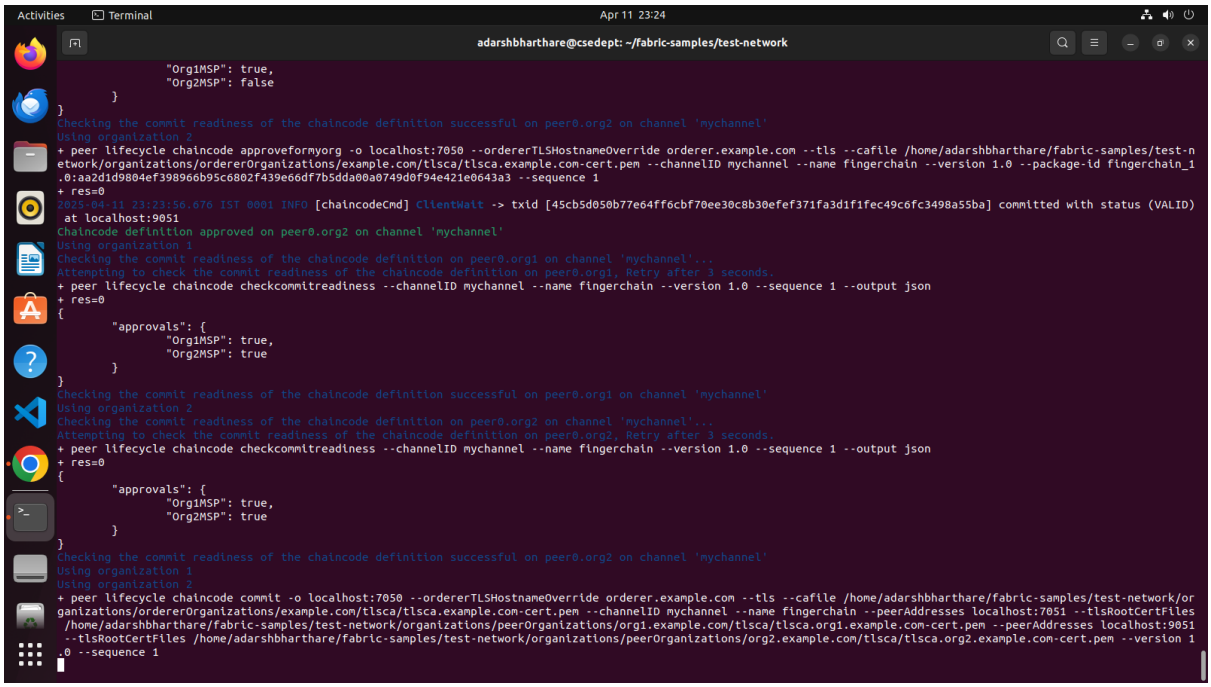


Figure 5: Figure depicts that both Org1 and Org2 approved chaincode fingerchain, version 1.0, sequence 1.

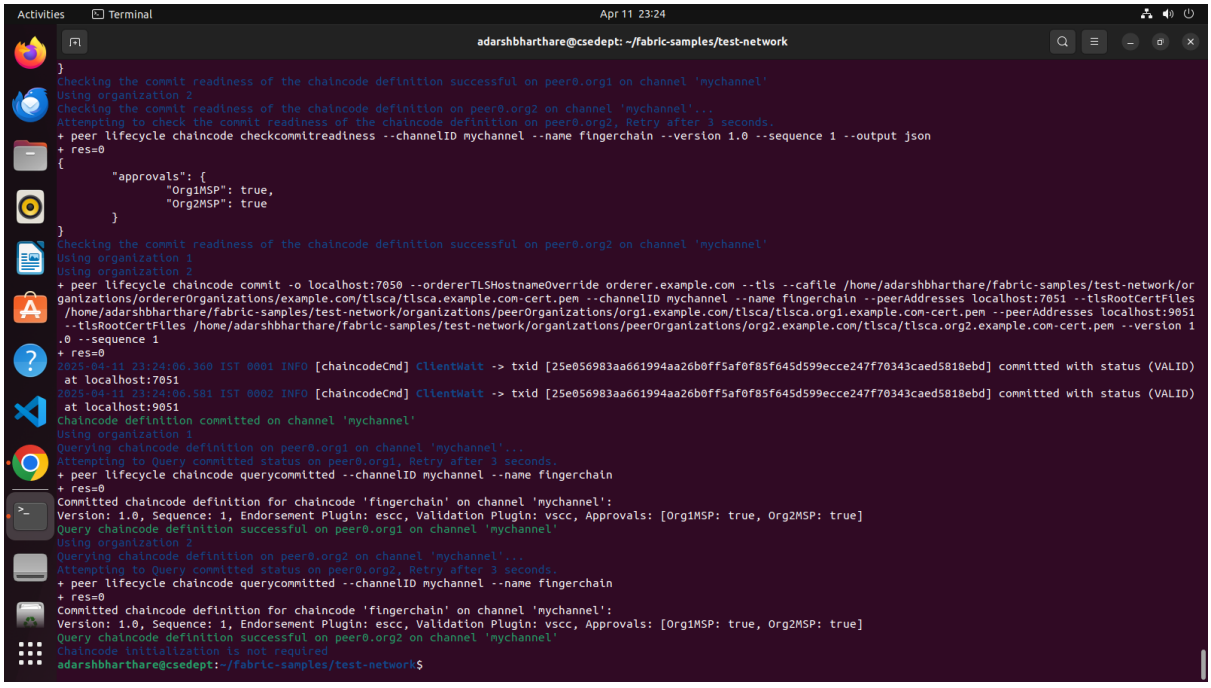


Figure 6: Chaincode Was Successfully Deployed on the Network

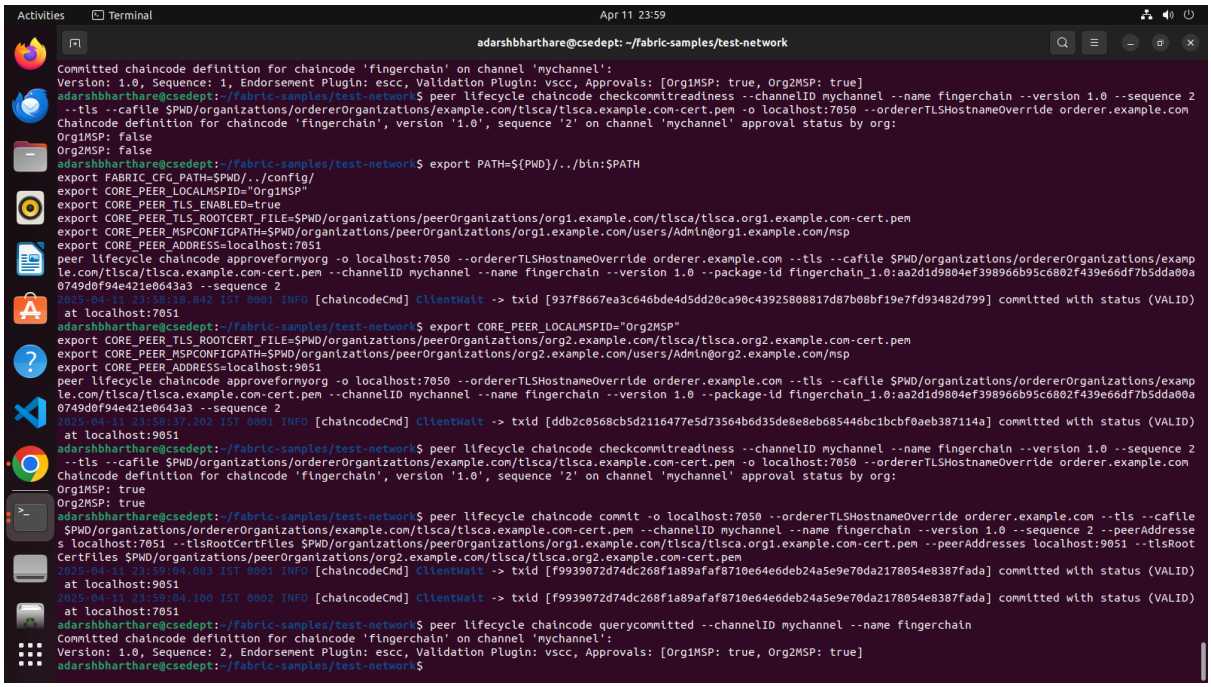


Figure 7: we're performing a chaincode upgrade from sequence 1 → sequence 2 for the chaincode fingerchain. In Fabric, every time we change the chaincode (code, endorsement policy, collections, etc.), we must increment the sequence number and re-approve the new definition by all organizations.

```
adarshbharthare@csedept:~/fingerchain-scripts$ cd ~/fingerchain-client
node app.js
cd ~/fingerchain-scripts
python3 ipfs_utils.py
python3 fingerprinting.py
Owner registered
User registered
Media uploaded with hash: QmbnnDCg6zzhmru8YMwv32oaTH8AaLLFtWMzC2fLV6PNMi
Media shared
Media info: {
  mediaID: 'media1',
  ownerID: 'owner1',
  title: 'Sample Media',
  hash: 'QmbnnDCg6zzhmru8YMwv32oaTH8AaLLFtWMzC2fLV6PNMi',
  price: 10,
  saleStatus: 'shared',
  sharedUsers: [ 'user1' ]
}
Run fingerprinting manually with: python3 ~/fingerchain-scripts/fingerprinting.py
IPFS Hash: QmX2Dr1WabW2GHQgH9NT8A3yKGk87yZDa1Z9s4t5dPuWKn
Downloaded: Test media content
Original fingerprint: [1 1 1 1 0 1 1 1 0 1]
Extracted fingerprint: [1 1 1 1 0 1 1 1 0 1]
adarshbharthare@csedept:~/fingerchain-scripts$
```

Figure 8: Here we've uploaded a media file to IPFS, registered the media and user, and successfully ran the fingerprinting process to generate and extract the fingerprint. The fingerprint represents a unique identifier for the media, which could be used for secure verification and matching in your FingerChain system.

Conclusion

a blockchain-based media sharing scheme tailored for multi-owner environments. By leveraging blockchain technology, media metadata aggregation is securely handled without the involvement of intermediaries. This decentralized approach empowers content owners with full control over their media, eliminating the need to rely on third-party platforms and enabling them to retain the entire value generated from their content.

To ensure secure and fair sharing, the system incorporates an asymmetric fingerprinting protocol that protects both the owner's copyright and the user's rights without depending on any trusted third party (TTP). Notably, the fingerprint embedding is performed on the user side, which significantly reduces the computational burden on the owner. Additionally, the system supports global media querying and robust user identity management, along with full traceability of media distribution and user interactions.

Despite these advancements, certain limitations remain. The challenge of ensuring fair exchange—guaranteeing that users receive valid media after payment—is not fully addressed. Moreover, the system does not yet account for the privacy concerns of owners regarding their sharing records. Addressing these gaps presents an avenue for future research. Future work will focus on designing a more comprehensive framework that supports fair trade mechanisms and enhances privacy-preserving features to better meet the practical needs of both content creators and users.

Bibliography

[1] Sixth Tone, "China's Web Fiction Writers Strike Over Copyright Confusion," `https://www.sixthtone.com/news/1005597`, 2020.

[2] Zeng et al., "A decentralized architecture of digital image copyright registration based on consortium blockchain," 2023.

[3] Agyekum et al., "Using perceptual hash to prevent duplicate registration of media copyright," 2023.

[4] Zhao et al., "Singular value decomposition for audio signal copyright registration," 2023.

[5] Xiao et al., "FingerChain: Copyrighted multi-owner media sharing," 2023.

[6] Meng et al., "Credible verification of image copyright using blockchain," 2023.

[7] Meng et al., "Blockchain-based image copyright verification," 2023.

[8] Li et al., "Blockchain watermarking scheme using compressed sensing," 2023.

[9] Natgunanathan et al., "Multi-layer watermarking embedding using blockchain," 2023.

[10] Ma et al., "Violation tracing in blockchain-based media sharing," 2023.

[11] Qureshi and Megías, "Homomorphic fingerprint embedding in blockchain," 2023.

[12] Frattolillo, "Fair watermarking protocol using blockchain," 2023.

[13] Zhao et al., "Copyright protection in music supply chain using vector quantization," 2023.

[14] Sheng et al., "TTP-free crowdsourcing data sharing using blockchain," 2023.

[15] Natgunanathan et al., "Multi-layer watermarking for content distribution," 2023.

[16] Ma et al., "Blockchain-based violation tracing," 2023.

[17] Qureshi and Megías, "Blockchain-based monitor for fingerprint generation," 2023.

[18] Frattolillo, "TTP-based fair watermarking protocol," 2023.

[19] Bianchi and Piva, "Asymmetric fingerprinting protocol for fair copyright protection," 2023.

[20] Other references [9–21], 2023.