A Decentralized Data Sharing Scheme for IPFS

Author：Linli Mo Supervisor：Jie Zhang

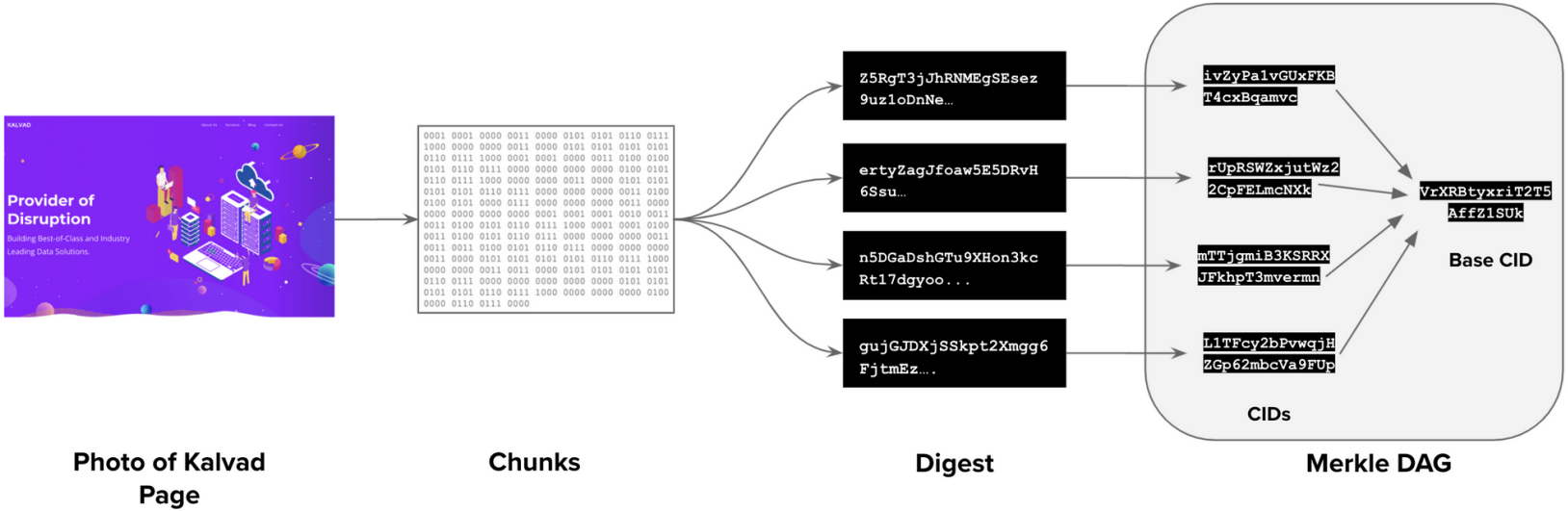
CONTENTS

01

1. Introduction

1. Introduction

- IPFS is changing the way we store and retrieve data in a decentralized way.
- But we need a secure, flexible, and decentralized way to share data on IPFS



- Link: https://blog.kalvad.com/myths-about-ipfs/

02

2. Related Work

2. Literature Review

- 1. Steichen at 2018 modified IPFS with **ACL** Ethereum smart contracts for efficient file sharing.

- 2. Sharma at 2021 added **ABE** technology to build a secure, efficient blockchain-based architecture for cloud storage systems.

- 3. BATTAH at 2020 proposed a decentralized blockchain system with **PRE** for multi-party access to encrypted IPFS data.

03

3. Design Goals

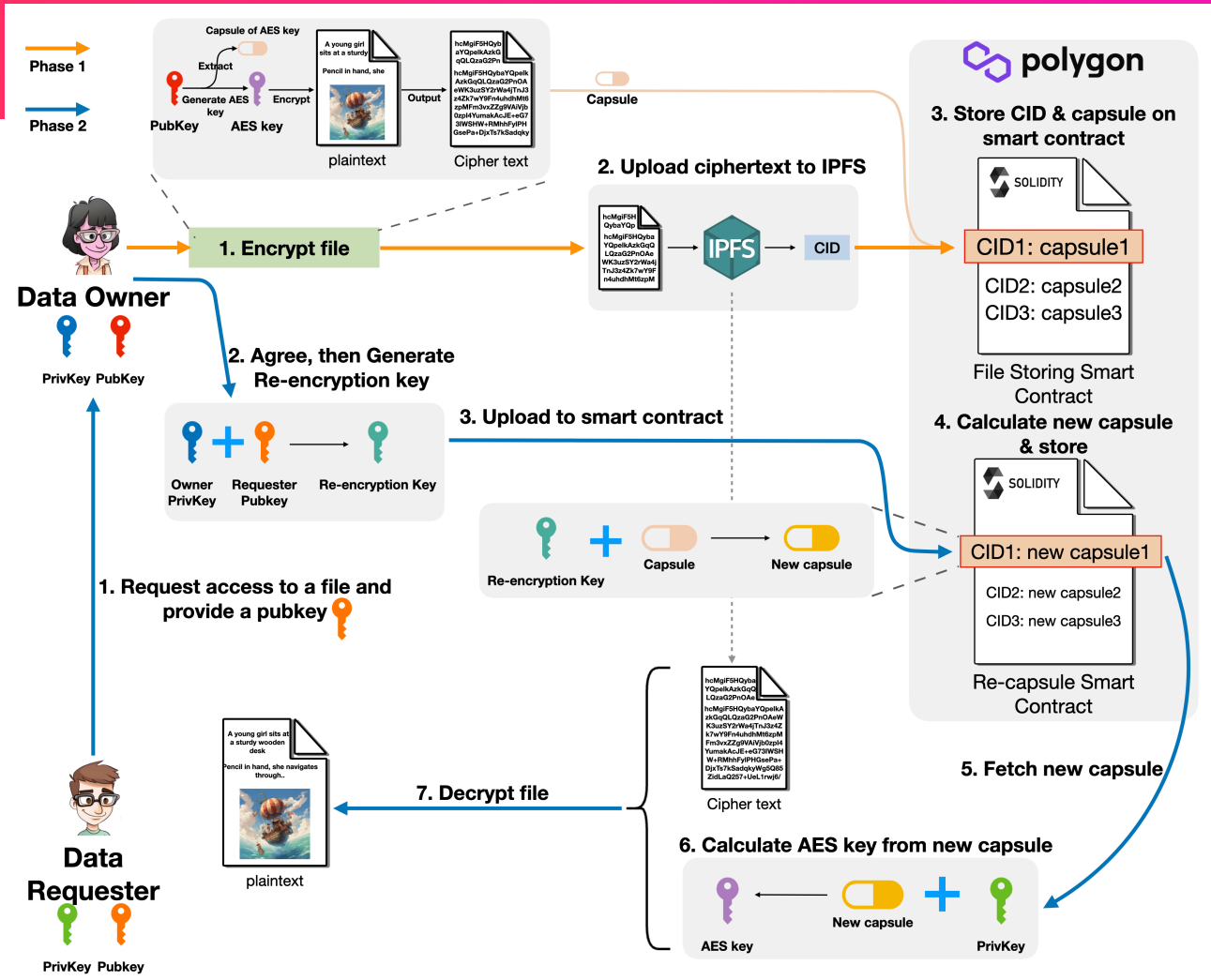
3. Design Goals

1. Data Confidentiality: Only the data owner and authorized data requester can access the content of files.

2. Efficiency and Cost-effectiveness: Core functions should be completed within a reasonable time.

04

4. System Overview


Phase 1
Phase 2
Capsule of AES key
Extract
Generate AES key
Encrypt
Output
PubKey
AES key
plaintext
Cipher text
Capsule
polygon
3. Store CID & capsule on smart contract
2. Upload ciphertext to IPFS
IPFS
CID
SOLIDITY
CID1: capsule1
CID2: capsule2
CID3: capsule3
File Storing Smart Contract
1. Encrypt file
Data Owner
PrivKey PubKey
2. Agree, then Generate Re-encryption key
Owner PrivKey
Requester Pubkey
Re-encryption Key
3. Upload to smart contract
4. Calculate new capsule & store
Re-encryption Key
Capsule
New capsule
CID1: new capsule1
CID2: new capsule2
CID3: new capsule3
Re-capsule Smart Contract
1. Request access to a file and provide a pubkey
5. Fetch new capsule
Cipher text
7. Decrypt file
plaintext
6. Calculate AES key from new capsule
New capsule
AES key
PrivKey
Data Requester
PrivKey Pubkey

05

5.1 File Encryption Algorithm

5.1 File Encryption Algorithm

Algorithm 1 File Encryption

Input: inputFilePath, outputFilePath, publickey
Output: CID, capsule

```
data ← ReadData(inputFilePath)
Generate AES key based on publickey:
    kp1 ← generate_key_pair()
    kp2 ← generate_key_pair()
    sk1 ← kp1.get_private_key()
    sk2 ← kp2.get_private_key()
    pk1 ← kp1.get_public_key()
    pk2 ← kp2.get_public_key()
    tmpHash ← [pk1, pk2]
    hash ← hash_to_scalar(tmpHash)
    partial_S ← sk1.add(sk2.mul(hash))
    pk_point ← publickey
    point_symmetric ← pk_point.mul(sk1.add(sk2))
    symmetric_key ← SHA256(point_symmetric)
encrypted_data ← Encrypt(data, symmetric_key)
Extract capsule information:
    E ← pk1
    V ← pk2
    S ← partial_S
    capsule ← newCapsule(E, V, S)
WriteData(outputFilePath, encrypted_data)
CID ← UploadToIPFS(outputFilePath)
return CID, capsule
```

05

5.2 Re-Encryption Key Calculation

5.2 Re-Encryption Key Calculation

Algorithm 3 Re-Encryption Key Calculation

Input: Owner privateKey, Requester publicKey
Output: re-encryption Key

1: $kp \leftarrow$ GenerateKeyPair()
2: $tmp\_sk \leftarrow kp$.get_private_key()
3: $tmp\_pk \leftarrow kp$.get_public_key()
4: $pk\_point \leftarrow (publicKey)$
5: $points\_for\_hash \leftarrow [tmp\_pk, pk\_point, pk\_point \times tmp\_sk]$
6: $hash \leftarrow$ hash_to_scalar($points\_for\_hash$)
7: $sk \leftarrow (privateKey)$
8: $hash\_inv \leftarrow$ inverse($hash$)
9: $rk \leftarrow sk \times hash\_inv$
10: $re\_key.\_private\_key \leftarrow rk$
11: $re\_key.\_public\_key \leftarrow tmp\_pk$
12: **return** re_key

05

5.3 Re-capsule Smart Contract

5.3 Re-capsule Smart Contract

Algorithm 4 Re-capsule Smart Contract

Input: capsule, re-encryption Key
Output: new capsule

1: $rk \leftarrow$ (re-encryption Key)
2: $new\_E \leftarrow$ capsule.E $\times$ rk._private_key
3: $new\_V \leftarrow$ capsule.V $\times$ rk._private_key
4: $new\_S \leftarrow$ capsule.S
5: $new\_ec\_point \leftarrow rk.\_public\_key$
6: $new\_capsule \leftarrow$ new Capsule(new_E, new_V, new_S, new_ec_point)
7: **return** $new\_capsule$

05

5.4 File Decryption Algorithm

5.4 File Decryption Algorithm

- Explain the File Decryption algorithm used in the file sharing phase.

Algorithm 5 File Decryption

Input: inputFilePath, outputFilePath, requester privateKey, new capsule
Output:

1: $data \leftarrow$ ReadData($inputFilePath$)
2: $priKey \leftarrow$ (requester privateKey)
3: $capsule \leftarrow$ (new capsule)
4: $new\_ECpoint \leftarrow$ capsule.new_ec_point
5: $new\_E \leftarrow$ capsule.new_E
6: $new\_V \leftarrow$ capsule.new_V
7: $public\_key \leftarrow$ priKey.public_key
8: $points\_for\_hash \leftarrow [new\_ECpoint, public\_key, new\_ECpoint \times priKey]$
9: $hash \leftarrow$ hash_to_scalar($points\_for\_hash$)
10: $tmp\_kdf\_point \leftarrow (new\_E + new\_V) \times hash$
11: $AES\_key \leftarrow$ SHA256(tmp_kdf_point)
12: $decrypted\_data \leftarrow$ decrypt(data, AES_Key)
13: WriteData($outputFilePath, decrypted\_data$)

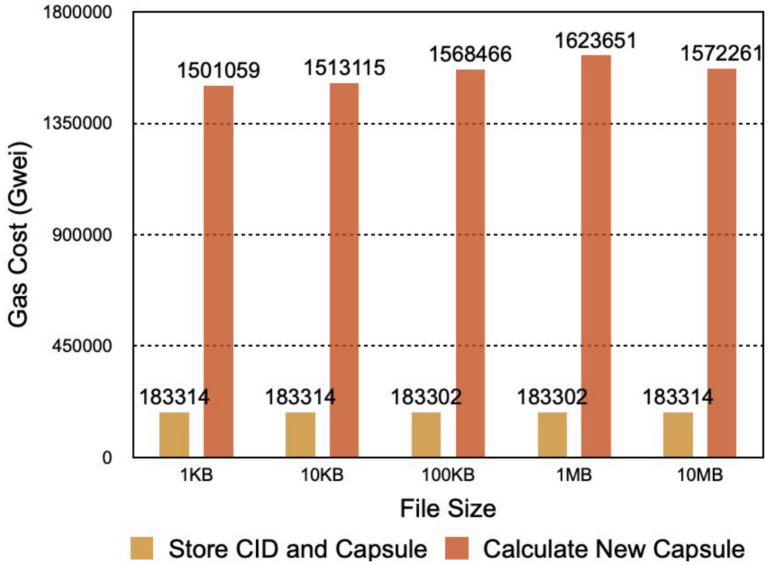
6

6. Experiment Results

6. Experiment Results

TABLE III
FUNCTIONALITY TESTS

Function	1KB	10KB	100KB	1MB	10MB
Encryption	✓	✓	✓	✓	✓
Proxy Re-Encryption	✓	✓	✓	✓	✓
Contract Integration	✓	✓	✓	✓	✓
File Retrieval	✓	✓	✓	✓	✓

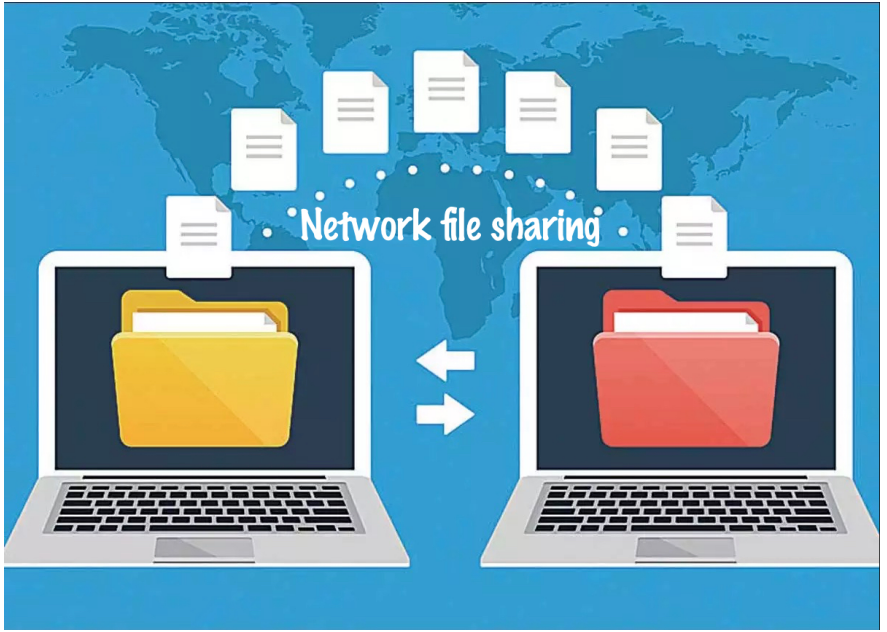


7

7. Future Work

7. Future Work

- Incorporation of time constraints within the proxy re-encryption scheme



- Link: https://zh.howtofix.guide/network-file-sharing/

8

8. Conclusion

8. Conclusion

- This work introduces a novel solution that combines blockchain technology, proxy re-encryption, and smart contracts to give a file sharing scheme for IPFS

THE END

THANKS