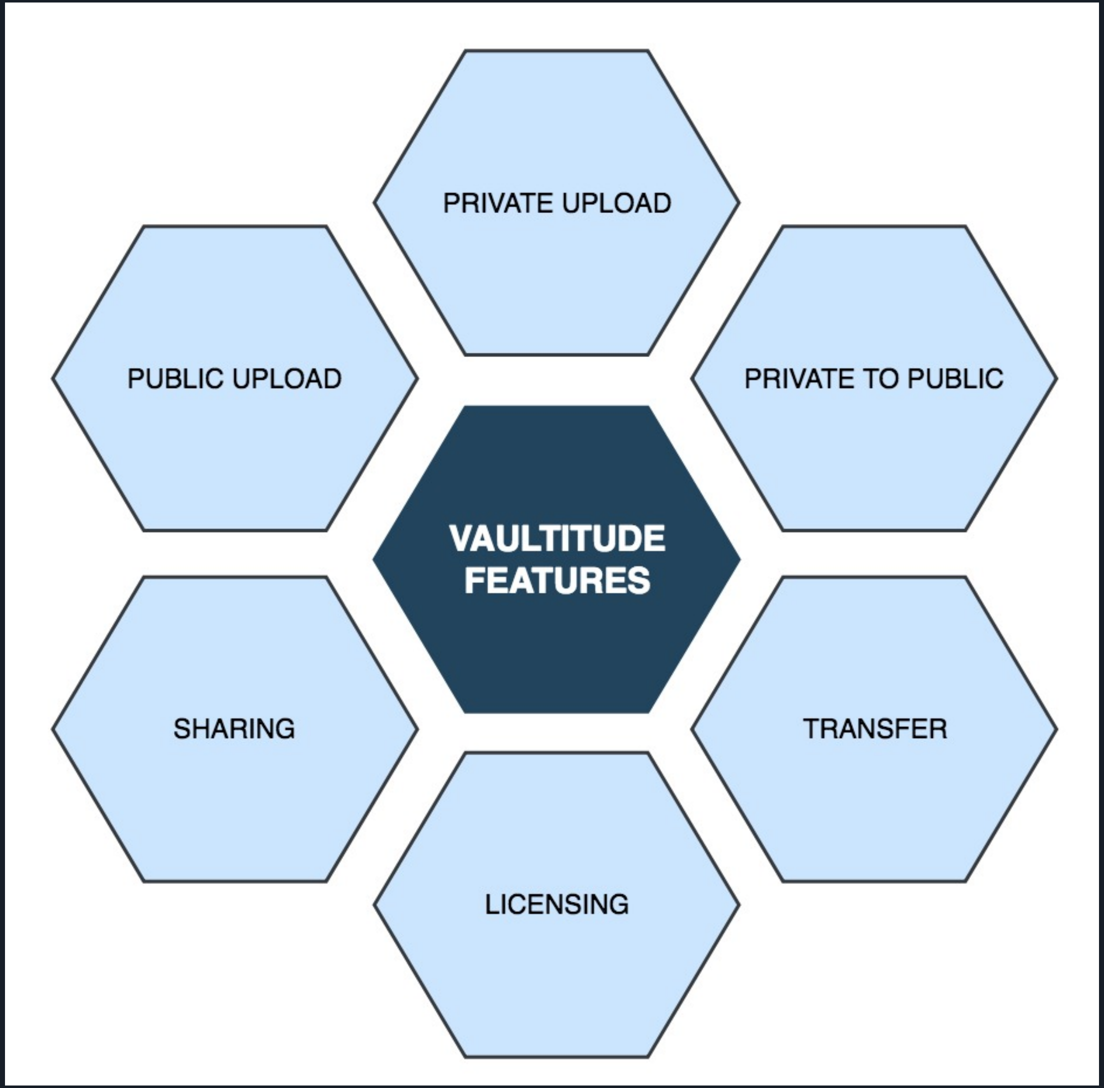
LimeChain

Telerik Academy

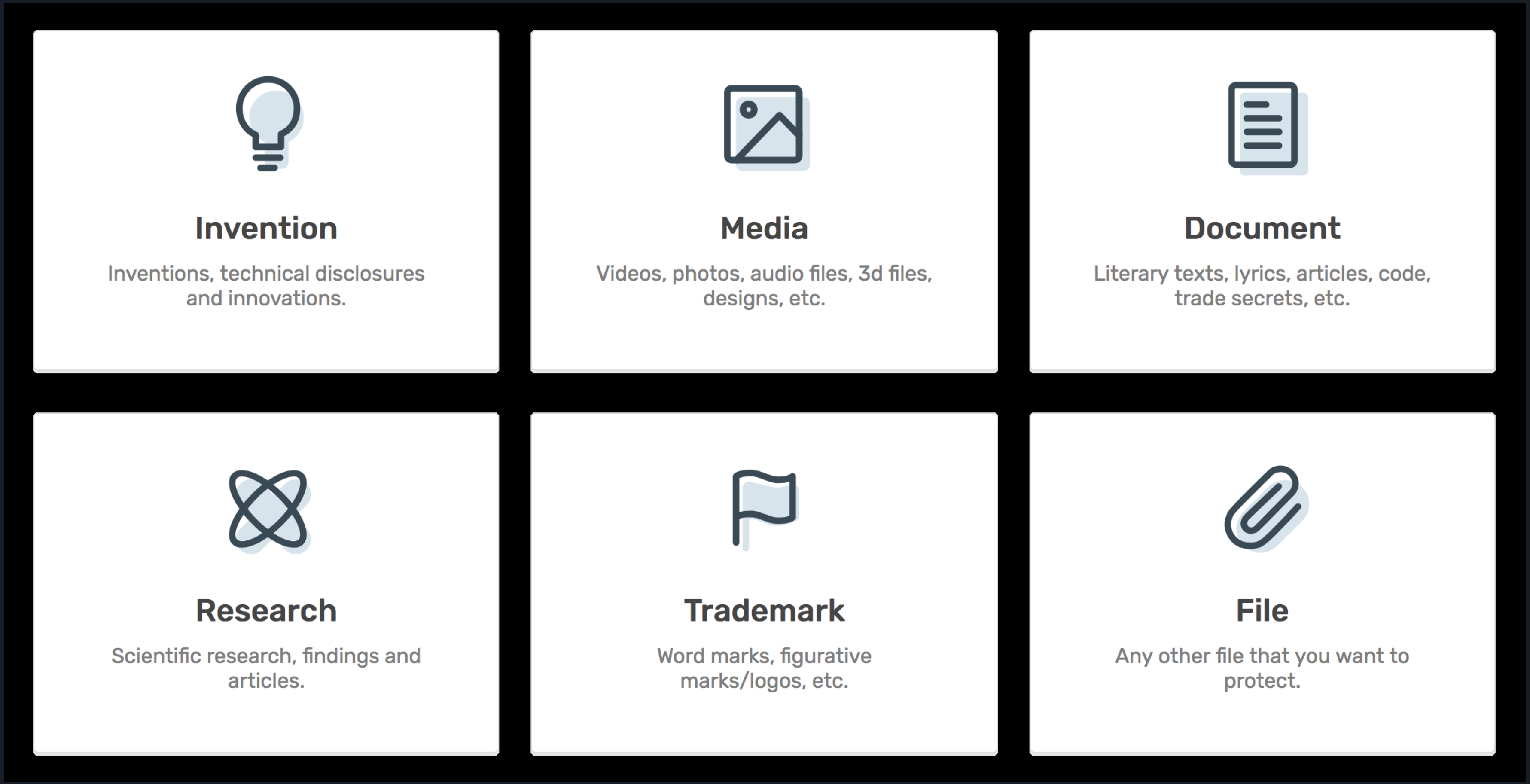
Cryptography, data encryption & IPFS

Lime Academy - Season 1

[Vaultitude] key features



[Vaultitude] IP types



Invention

Inventions, technical disclosures and innovations.

Media

Videos, photos, audio files, 3d files, designs, etc.

Document

Literary texts, lyrics, articles, code, trade secrets, etc.

Research

Scientific research, findings and articles.

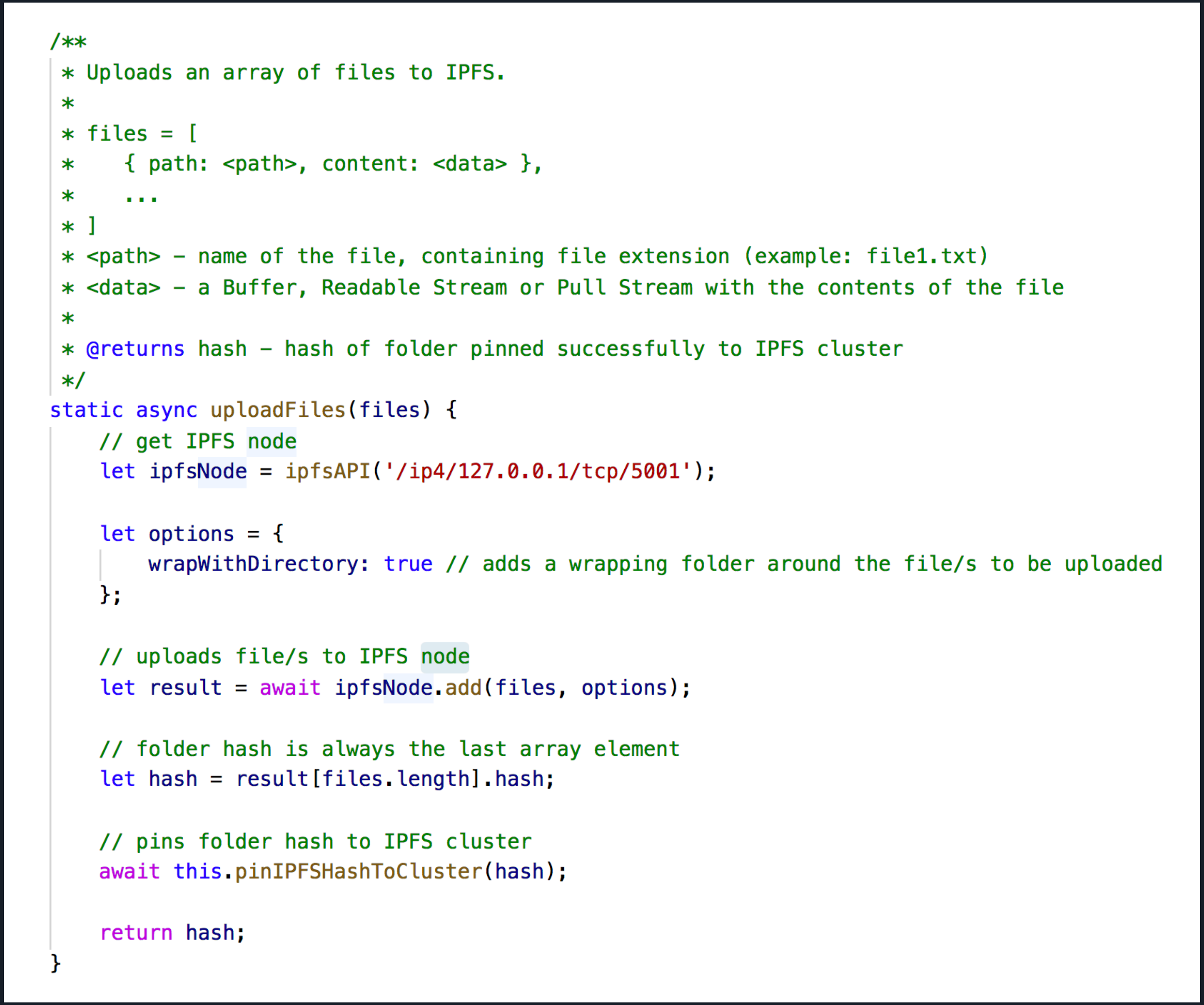
Trademark

Word marks, figurative marks/logos, etc.

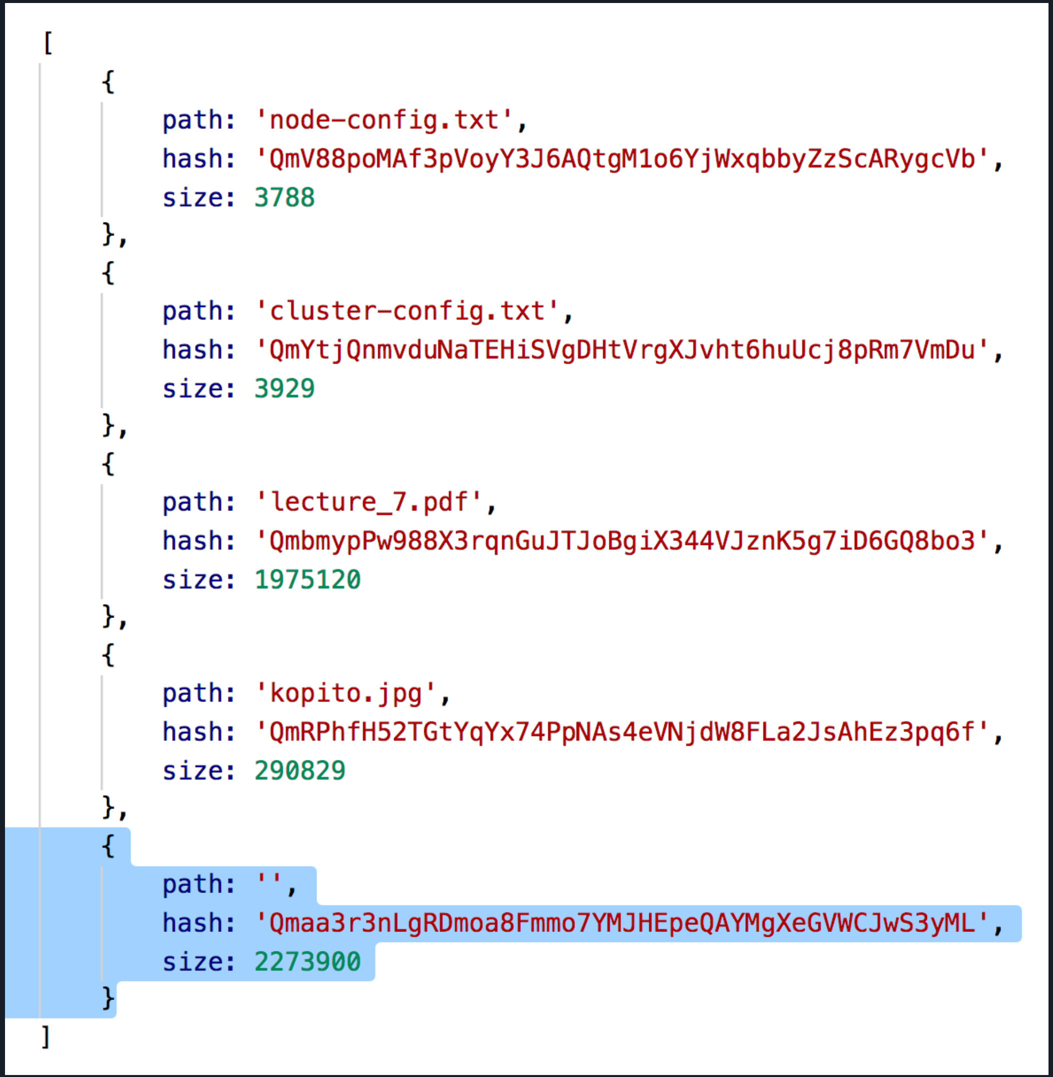
File

Any other file that you want to protect.

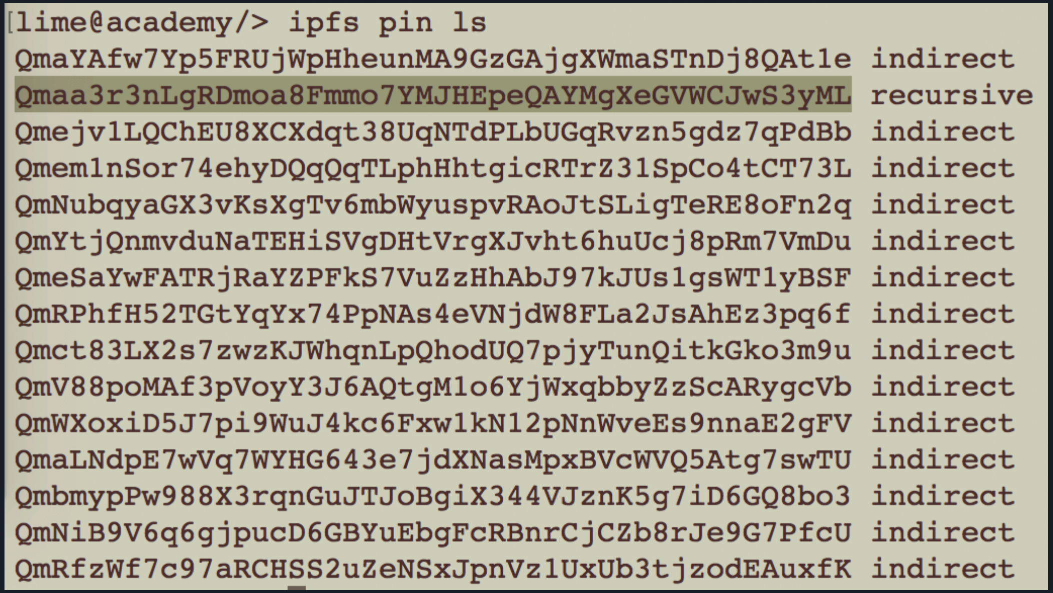
Upload public IP



```
/**
 * Uploads an array of files to IPFS.
 *
 * files = [
 *    { path: <path>, content: <data> },
 *    ...
 * ]
 * <path> - name of the file, containing file extension (example: file1.txt)
 * <data> - a Buffer, Readable Stream or Pull Stream with the contents of the file
 *
 * @returns hash - hash of folder pinned successfully to IPFS cluster
 */
static async uploadFiles(files) {
    // get IPFS node
    let ipfsNode = ipfsAPI('/ip4/127.0.0.1/tcp/5001');

    let options = {
        wrapWithDirectory: true // adds a wrapping folder around the file/s to be uploaded
    };

    // uploads file/s to IPFS node
    let result = await ipfsNode.add(files, options);

    // folder hash is always the last array element
    let hash = result[files.length].hash;

    // pins folder hash to IPFS cluster
    await this.pinIPFSHashToCluster(hash);

    return hash;
}
```

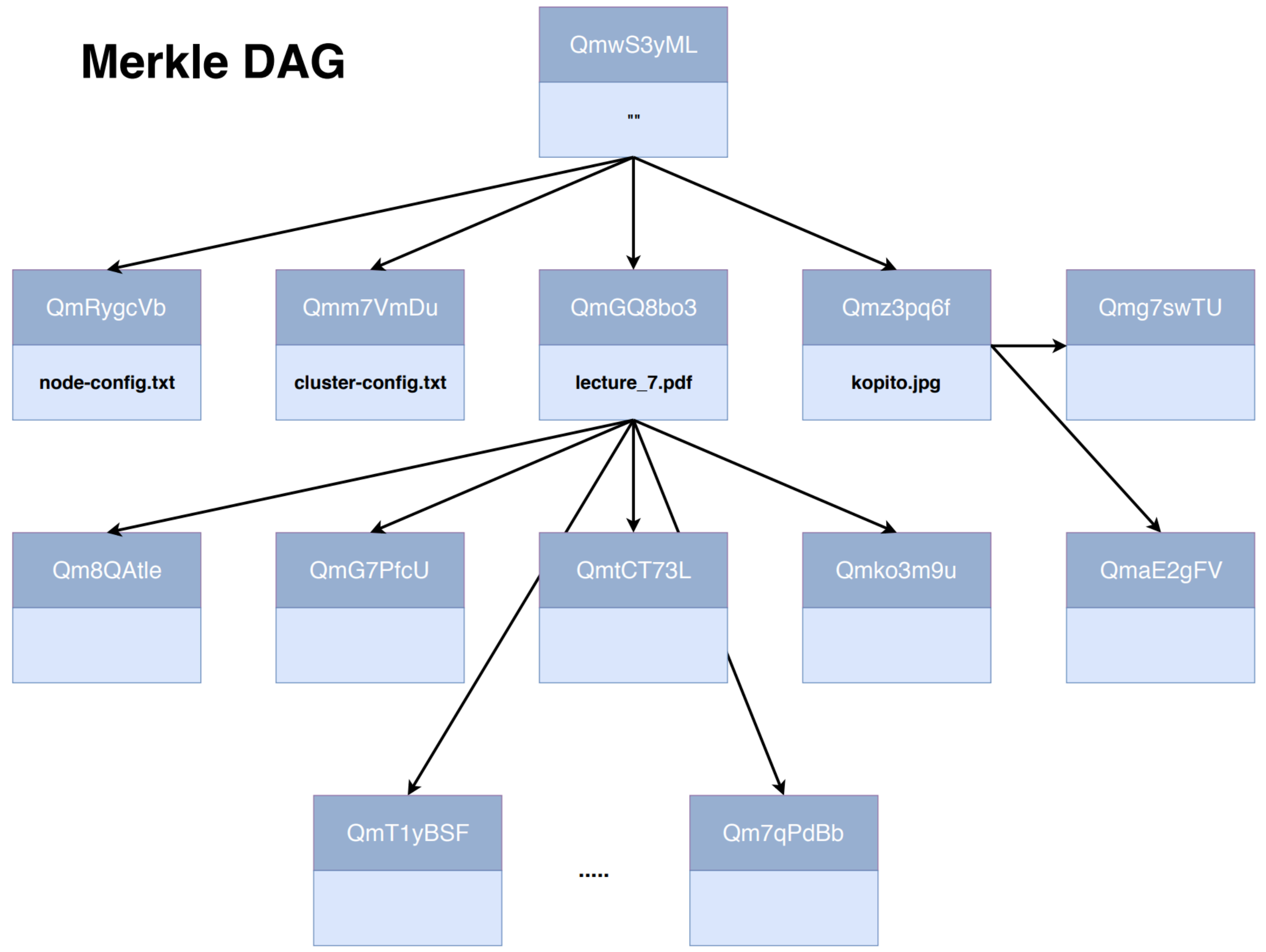


```
[
    {
        path: 'node-config.txt',
        hash: 'QmV88poMAf3pVoyY3J6AQtgM1o6YjWxqbbyZzScARygcVb',
        size: 3788
    },
    {
        path: 'cluster-config.txt',
        hash: 'QmYtjQnmvduNaTEHiSVgDHtVrgXJvht6huUcj8pRm7VmDu',
        size: 3929
    },
    {
        path: 'lecture_7.pdf',
        hash: 'QmbmypPw988X3rqnGuJTJoBgiX344VJznK5g7iD6GQ8bo3',
        size: 1975120
    },
    {
        path: 'kopito.jpg',
        hash: 'QmRPhfH52TGtYqYx74PpNAs4eVNjdW8FLa2JsAhEz3pq6f',
        size: 290829
    },
    {
        path: '',
        hash: 'Qmaa3r3nLgRDmoa8Fmmo7YMJHEpeQAYMgXeGVWCJwS3yML',
        size: 2273900
    }
]
```



```
[lime@academy/> ipfs pin ls
QmaYAfw7Yp5FRUjWpHheunMA9GzGAjgXWmaSTnDj8QAt1e indirect
Qmaa3r3nLgRDmoa8Fmmo7YMJHEpeQAYMgXeGVWCJwS3yML recursive
Qmejv1LQChEU8XCXdqt38UqNTdPLbUGqRvzn5gdz7qPdBb indirect
Qmem1nSor74ehyDQqQqTLphHhtgicRTrZ31SpCo4tCT73L indirect
QmNubqyaGX3vKsXgTv6mbWyuspvRAoJtSLigTeRE8oFn2q indirect
QmYtjQnmvduNaTEHiSVgDHtVrgXJvht6huUcj8pRm7VmDu indirect
QmeSaYwFATRjRaYZPFkS7VuZzHhAbJ97kJUs1gsWT1yBSF indirect
QmRPhfH52TGtYqYx74PpNAs4eVNjdW8FLa2JsAhEz3pq6f indirect
Qmct83LX2s7zwzKJWhqnLpQhodUQ7pjyTunQitkGko3m9u indirect
QmV88poMAf3pVoyY3J6AQtgM1o6YjWxqbbyZzScARygcVb indirect
QmWXoxiD5J7pi9WuJ4kc6Fxw1kN12pNnWveEs9nnaE2gFV indirect
QmaLNdpE7wVq7WYHG643e7jdXNasMpxBVcWVQ5Atg7swTU indirect
QmbmypPw988X3rqnGuJTJoBgiX344VJznK5g7iD6GQ8bo3 indirect
QmNiB9V6q6gjpucD6GBYuEbgFcRBnrCjCZb8rJe9G7PfcU indirect
QmRfzWf7c97aRCHSS2uZeNSxJpnVz1UxUb3tjzodEAuxfK indirect
```

Merkle DAG


QmwS3yML
""
QmRygcVb
node-config.txt
Qmm7VmDu
cluster-config.txt
QmGQ8bo3
lecture_7.pdf
Qmz3pq6f
kopito.jpg
Qmg7swTU
Qm8QAtle
QmG7PfcU
QmtCT73L
Qmko3m9u
QmaE2gFV
QmT1yBSF
.....
Qm7qPdBb

IPFS hash format

Multihash (base58 encoded) - CID v.0

Qmaa3r3nLgRDmoa8Fmmo7YMJHEpeQAYMgXeGVWCJwS3yML

Q => SHA-256
m => length (32 bytes)
aa3r3nLgRDmoa8Fmmo7YMJHEpeQAYMgXeGVWCJwS3yML => hash

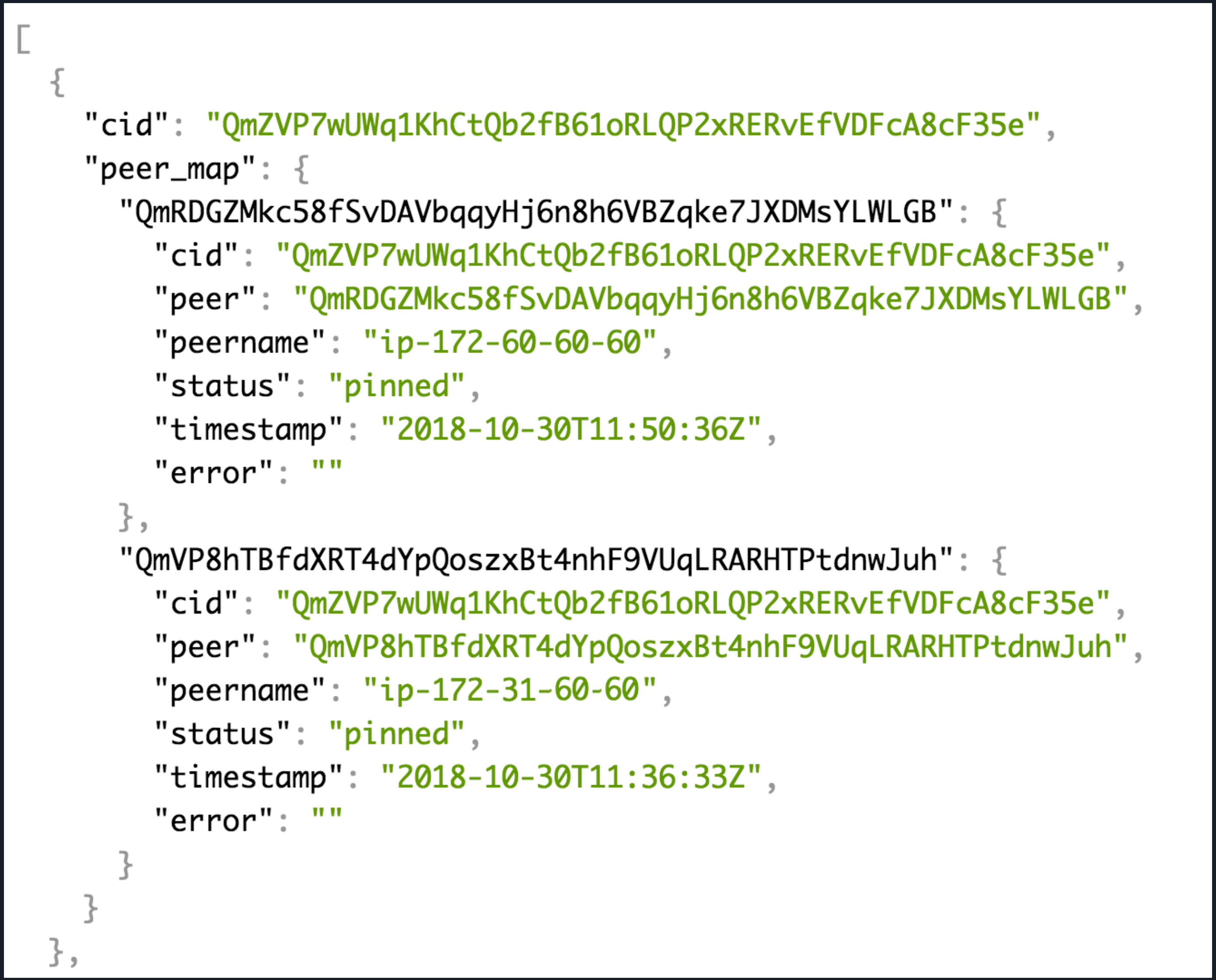
Convert IPFS hash to bytes32:

```
'0x' + bs58.decode(hash).slice(2).toString('hex')
```

Convert bytes32 to IPFS hash:

```
bs58.encode(Buffer.from('1220' + short.slice(2), ‘hex'));
```

Pinning

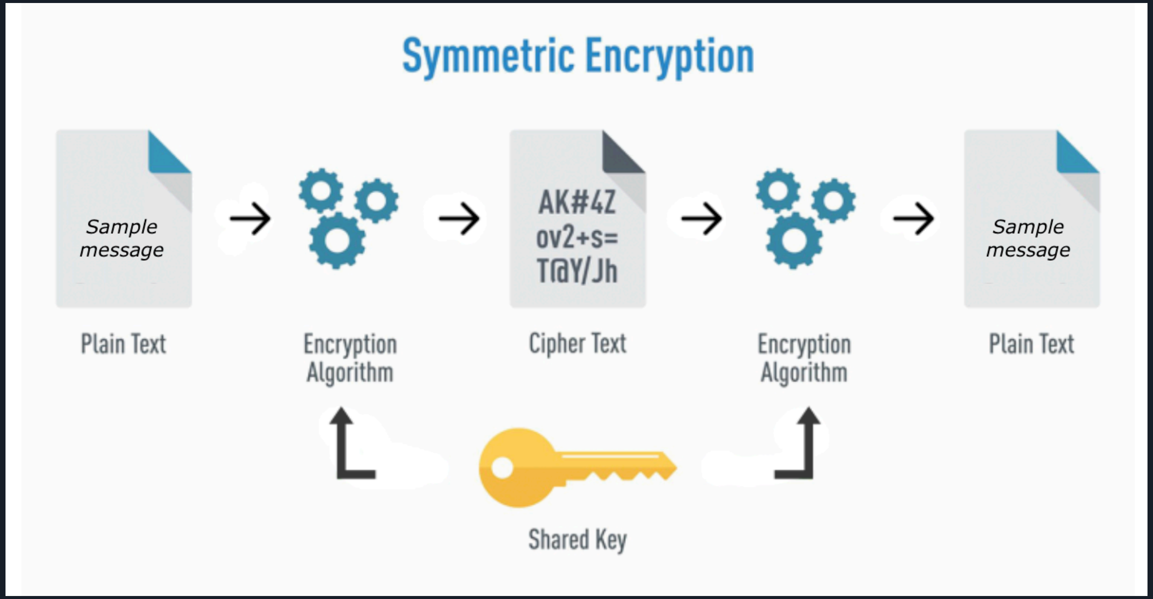


```
[
  {
    "cid": "QmZVP7wUWq1KhCtQb2fB61oRLQP2xRERvEfVDFcA8cF35e",
    "peer_map": {
      "QmRDGZMkc58fSvDAVbqqyHj6n8h6VBZqke7JXDMsYLWLGB": {
        "cid": "QmZVP7wUWq1KhCtQb2fB61oRLQP2xRERvEfVDFcA8cF35e",
        "peer": "QmRDGZMkc58fSvDAVbqqyHj6n8h6VBZqke7JXDMsYLWLGB",
        "peername": "ip-172-60-60-60",
        "status": "pinned",
        "timestamp": "2018-10-30T11:50:36Z",
        "error": ""
      },
      "QmVP8hTBfdXRT4dYpQoszxBt4nhF9VUqLRARHTPtdnwJuh": {
        "cid": "QmZVP7wUWq1KhCtQb2fB61oRLQP2xRERvEfVDFcA8cF35e",
        "peer": "QmVP8hTBfdXRT4dYpQoszxBt4nhF9VUqLRARHTPtdnwJuh",
        "peername": "ip-172-31-60-60",
        "status": "pinned",
        "timestamp": "2018-10-30T11:36:33Z",
        "error": ""
      }
    }
  },
```

Upload private IP

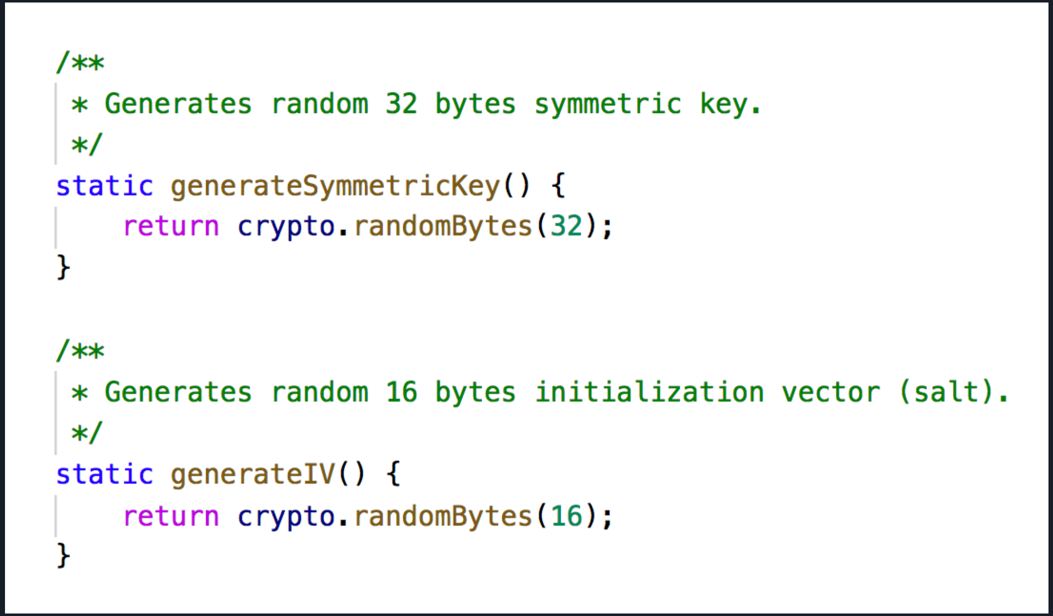
IP content encryption (crypto module)

- private IP could contain big files
- fast encryption is required
- 1 IP/key … N users with access

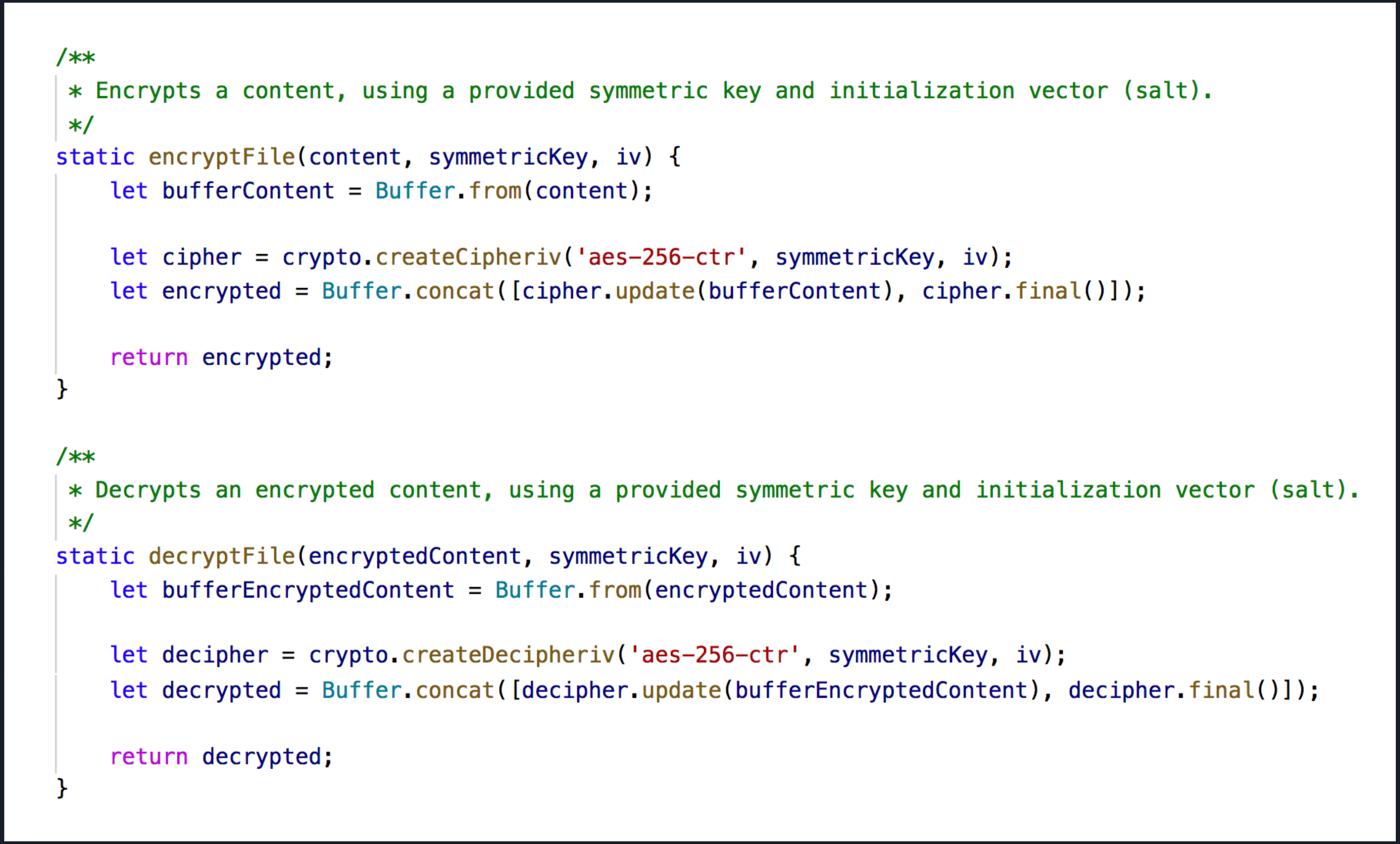


Adding randomness with initialization vector (salt)

- stronger encryption
- encrypted output is always different
- prevents dictionary, rainbow, brute-force attacks
- generate a unique random IV per encryption run



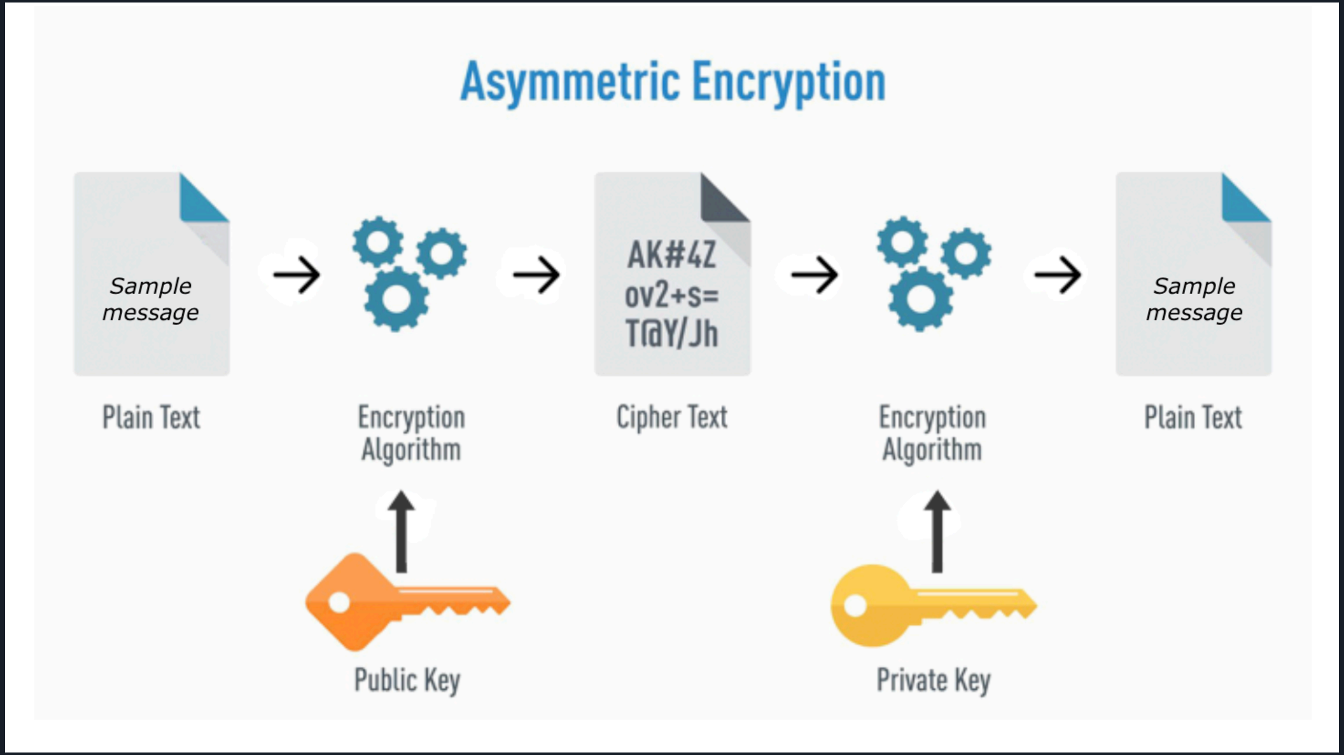
```
/**
 * Generates random 32 bytes symmetric key.
 */
static generateSymmetricKey() {
    return crypto.randomBytes(32);
}

/**
 * Generates random 16 bytes initialization vector (salt).
 */
static generateIV() {
    return crypto.randomBytes(16);
}
```

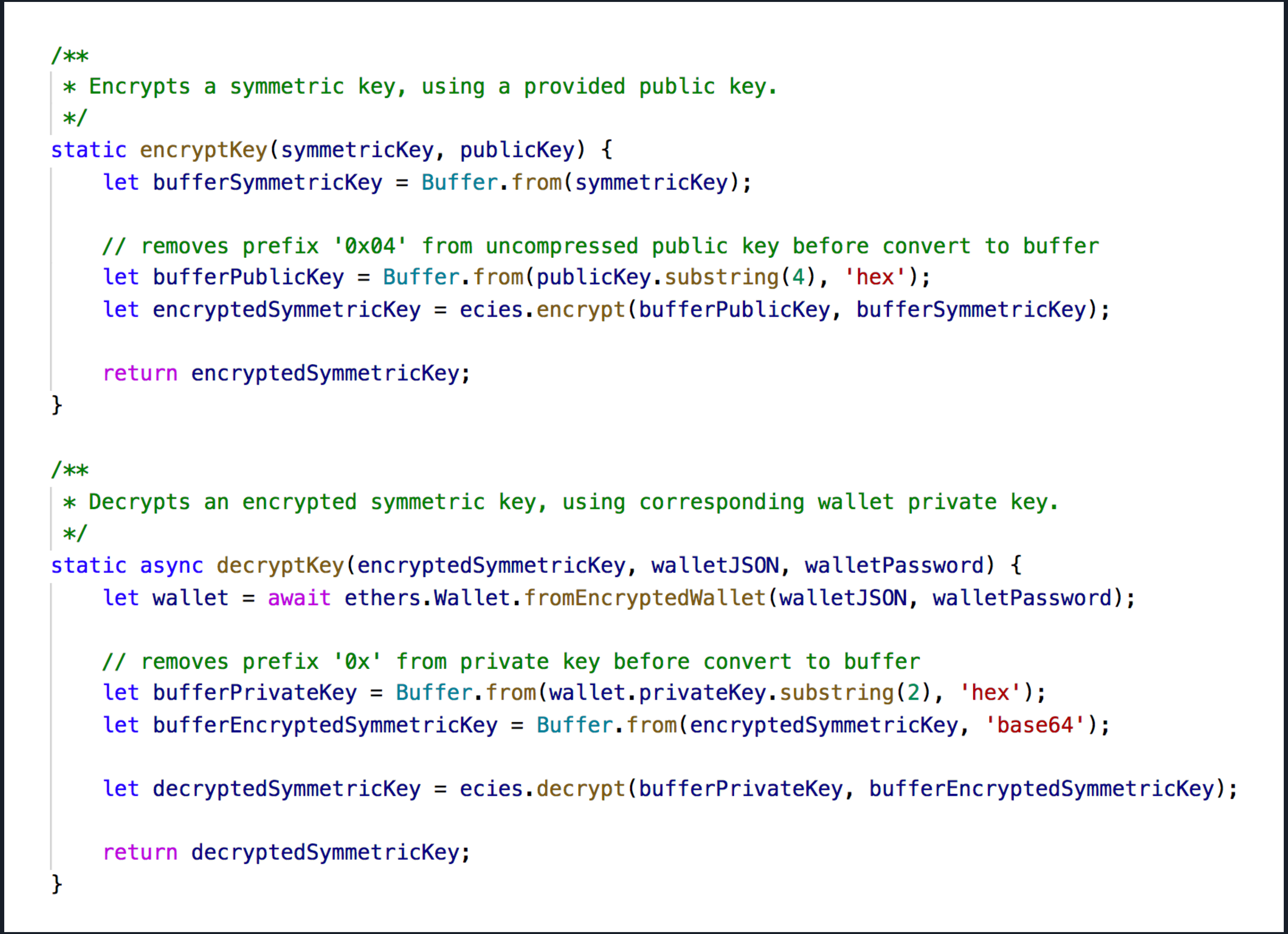


```
/**
 * Encrypts a content, using a provided symmetric key and initialization vector (salt).
 */
static encryptFile(content, symmetricKey, iv) {
    let bufferContent = Buffer.from(content);

    let cipher = crypto.createCipheriv('aes-256-ctr', symmetricKey, iv);
    let encrypted = Buffer.concat([cipher.update(bufferContent), cipher.final()]);

    return encrypted;
}

/**
 * Decrypts an encrypted content, using a provided symmetric key and initialization vector (salt).
 */
static decryptFile(encryptedContent, symmetricKey, iv) {
    let bufferEncryptedContent = Buffer.from(encryptedContent);

    let decipher = crypto.createDecipheriv('aes-256-ctr', symmetricKey, iv);
    let decrypted = Buffer.concat([decipher.update(bufferEncryptedContent), decipher.final()]);

    return decrypted;
}
```

Symmetric key exchange (eth-ecies module)

- single key => multiple recipients (key per IP)
- key is encrypted with public key of each recipients wallet
- user must have access only to his own key

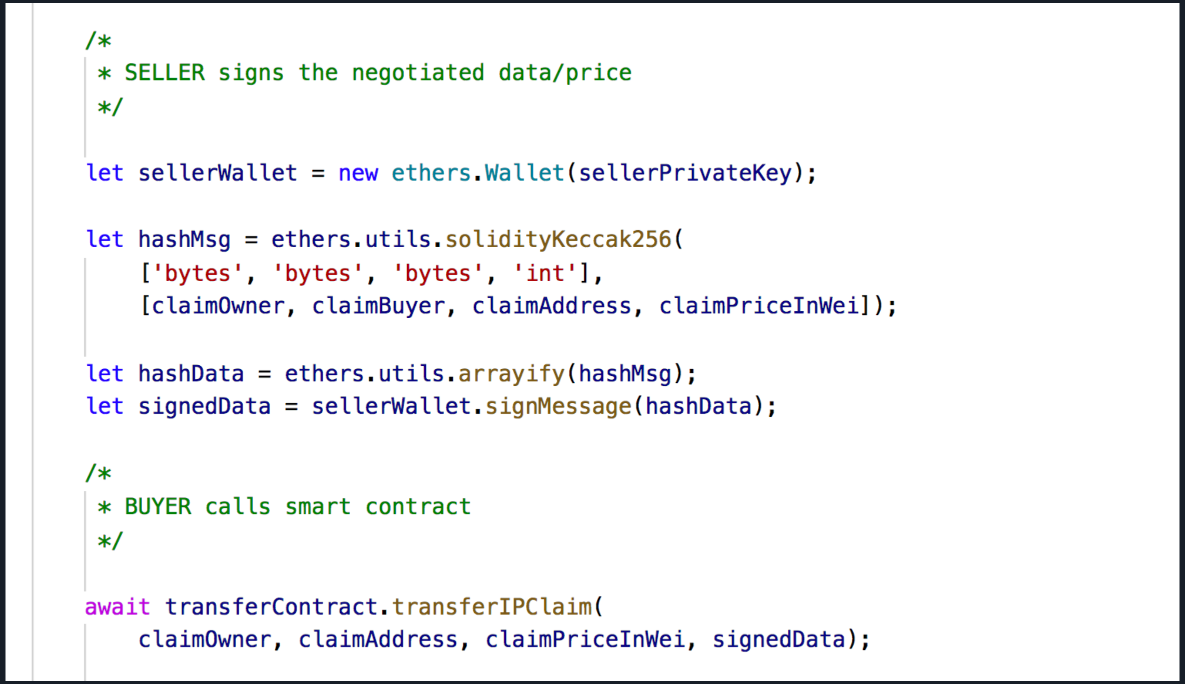




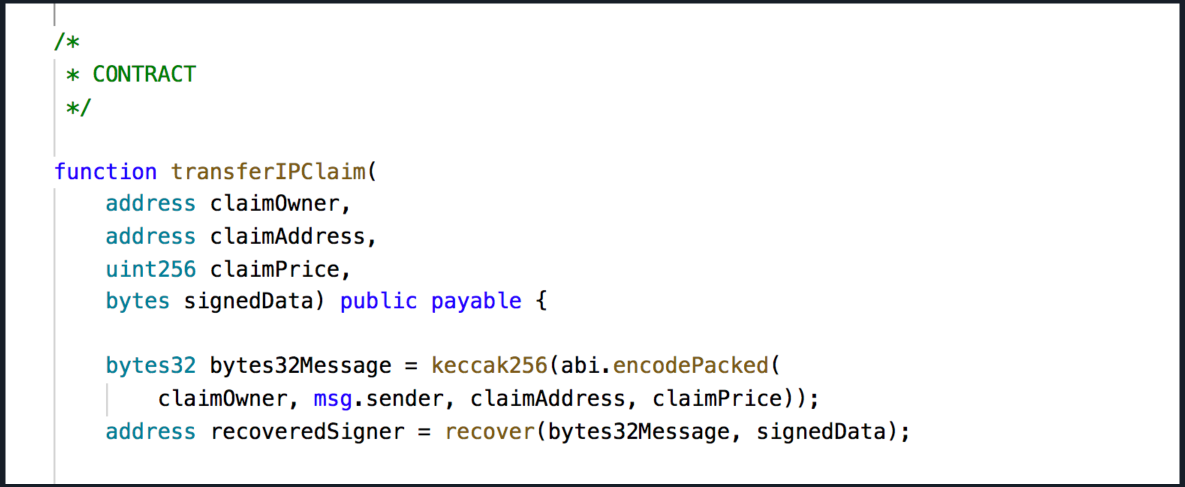
```javascript
/**
 * Encrypts a symmetric key, using a provided public key.
 */
static encryptKey(symmetricKey, publicKey) {
    let bufferSymmetricKey = Buffer.from(symmetricKey);

    // removes prefix '0x04' from uncompressed public key before convert to buffer
    let bufferPublicKey = Buffer.from(publicKey.substring(4), 'hex');
    let encryptedSymmetricKey = ecies.encrypt(bufferPublicKey, bufferSymmetricKey);

    return encryptedSymmetricKey;
}

/**
 * Decrypts an encrypted symmetric key, using corresponding wallet private key.
 */
static async decryptKey(encryptedSymmetricKey, walletJSON, walletPassword) {
    let wallet = await ethers.Wallet.fromEncryptedWallet(walletJSON, walletPassword);

    // removes prefix '0x' from private key before convert to buffer
    let bufferPrivateKey = Buffer.from(wallet.privateKey.substring(2), 'hex');
    let bufferEncryptedSymmetricKey = Buffer.from(encryptedSymmetricKey, 'base64');

    let decryptedSymmetricKey = ecies.decrypt(bufferPrivateKey, bufferEncryptedSymmetricKey);

    return decryptedSymmetricKey;
}
```

Transfer IP

ECRecover



```
/*
 * SELLER signs the negotiated data/price
 */

let sellerWallet = new ethers.Wallet(sellerPrivateKey);

let hashMsg = ethers.utils.solidityKeccak256(
    ['bytes', 'bytes', 'bytes', 'int'],
    [claimOwner, claimBuyer, claimAddress, claimPriceInWei]);

let hashData = ethers.utils.arrayify(hashMsg);
let signedData = sellerWallet.signMessage(hashData);

/*
 * BUYER calls smart contract
 */

await transferContract.transferIPClaim(
    claimOwner, claimAddress, claimPriceInWei, signedData);
```



```
/*
 * CONTRACT
 */

function transferIPClaim(
    address claimOwner,
    address claimAddress,
    uint256 claimPrice,
    bytes signedData) public payable {

    bytes32 bytes32Message = keccak256(abi.encodePacked(
        claimOwner, msg.sender, claimAddress, claimPrice));
    address recoveredSigner = recover(bytes32Message, signedData);
```

Homework

- Encrypt a bunch of files (one of them should be 50 MB) and upload them together in a folder to local IPFS node.
- Retrieve encrypted content from IPFS and successfully decrypt it.
- Try first with symmetric and then with asymmetric encryption.
- Check the difference in time (encryption/decryption processing)

Further reading

- crypto module
  https://nodejs.org/api/crypto.html
- eth-ecies module
  https://github.com/libertylocked/eth-ecies
- Stronger encryption and decryption in Node.js
  http://vancelucas.com/blog/stronger-encryption-and-decryption-in-node-js/
- Encrypt and decrypt content in Node.js
  https://lollyrock.com/articles/nodejs-encryption/
- IPFS publishing and IP cluster
  https://medium.com/mvp-workshop/ipfs-publishing-and-ipfs-cluster-cff3a099993a

Q & A

Thanks !

vlado@limechain.tech