CloudChain: implementation of a Blockchain-based Flight Data Recorder for Cloud Accountability

Paola Persico

paola.persico@studio.unibo.it

Magistrale in Informatica

2020/2021

Introduction

- **Cloud computing** → computing resources as services

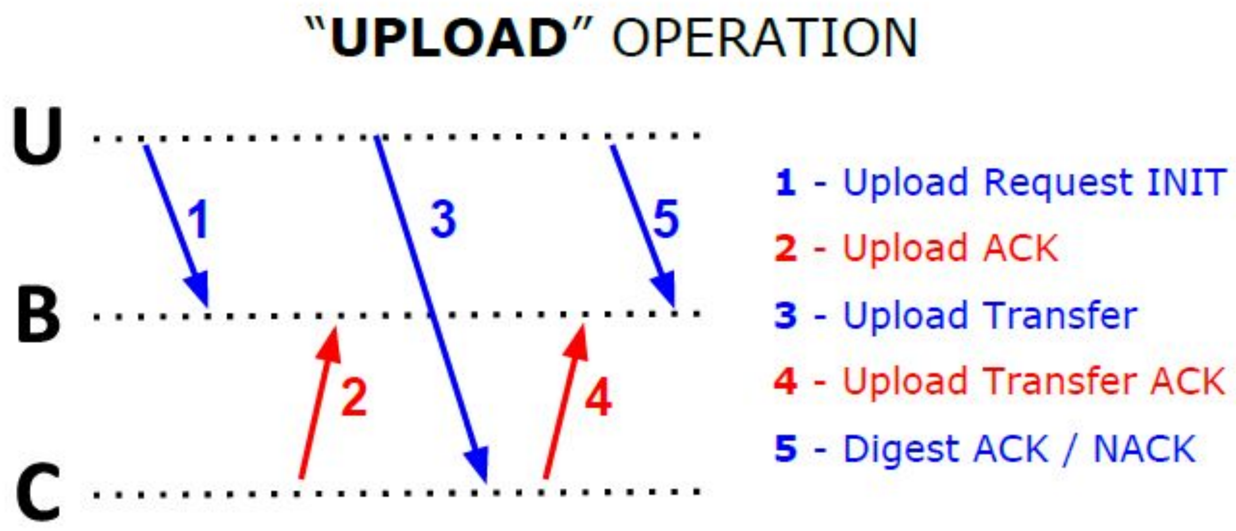
- Problems:
  - SLA violations
  - Security violations
  - Data corruption
  - Data leakage

- Goal: **accountability**
  - completeness
  - accuracy
  - third-party verifiability

Introduction

- Solution:
  1. **blockchain-based event logging**
     a. tamper-proof
     b. pseudo-anonymous
  2. **smart contract**
     a. SLA verificator
     b. accessible by every involved entity
     c. automatic compensation
- Case study: **cloud storage**
  - File upload
  - File delete
  - File read

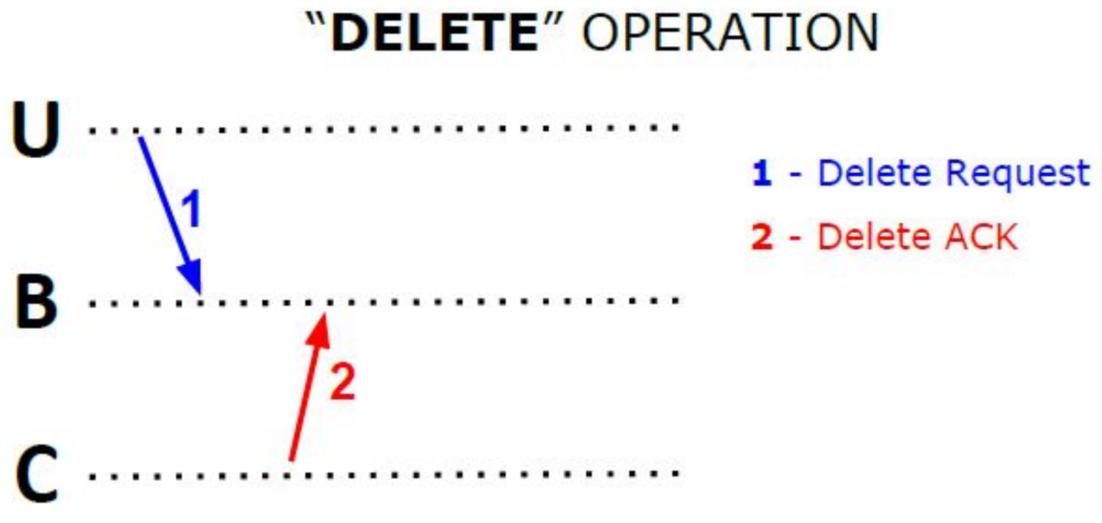
Protocol design: Upload



- ❑ File encryption before upload
- ❑ File digest computation and storage after upload
- ❑ File deletion if digest NACK

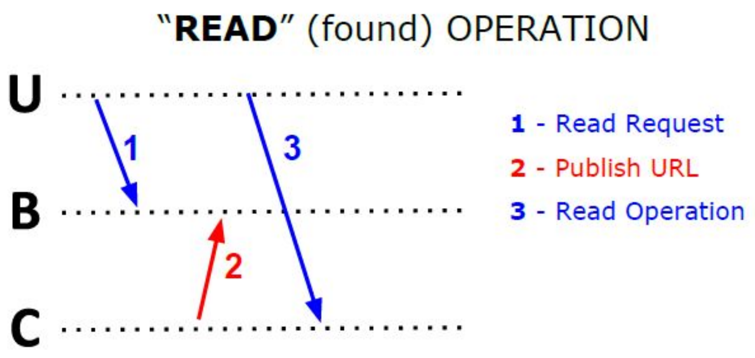
- ❖ Possible SLA violation
  - ➢ file is not deleted after digest NACK

Protocol design: Delete

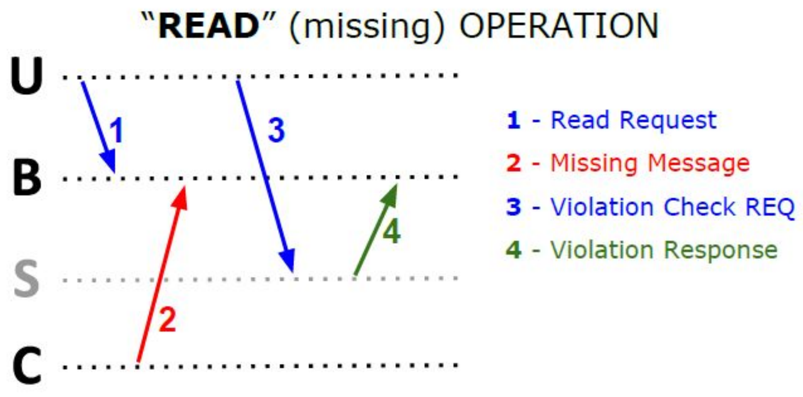


- Possible SLA violation
  - file is not deleted after delete request

Protocol design: Read



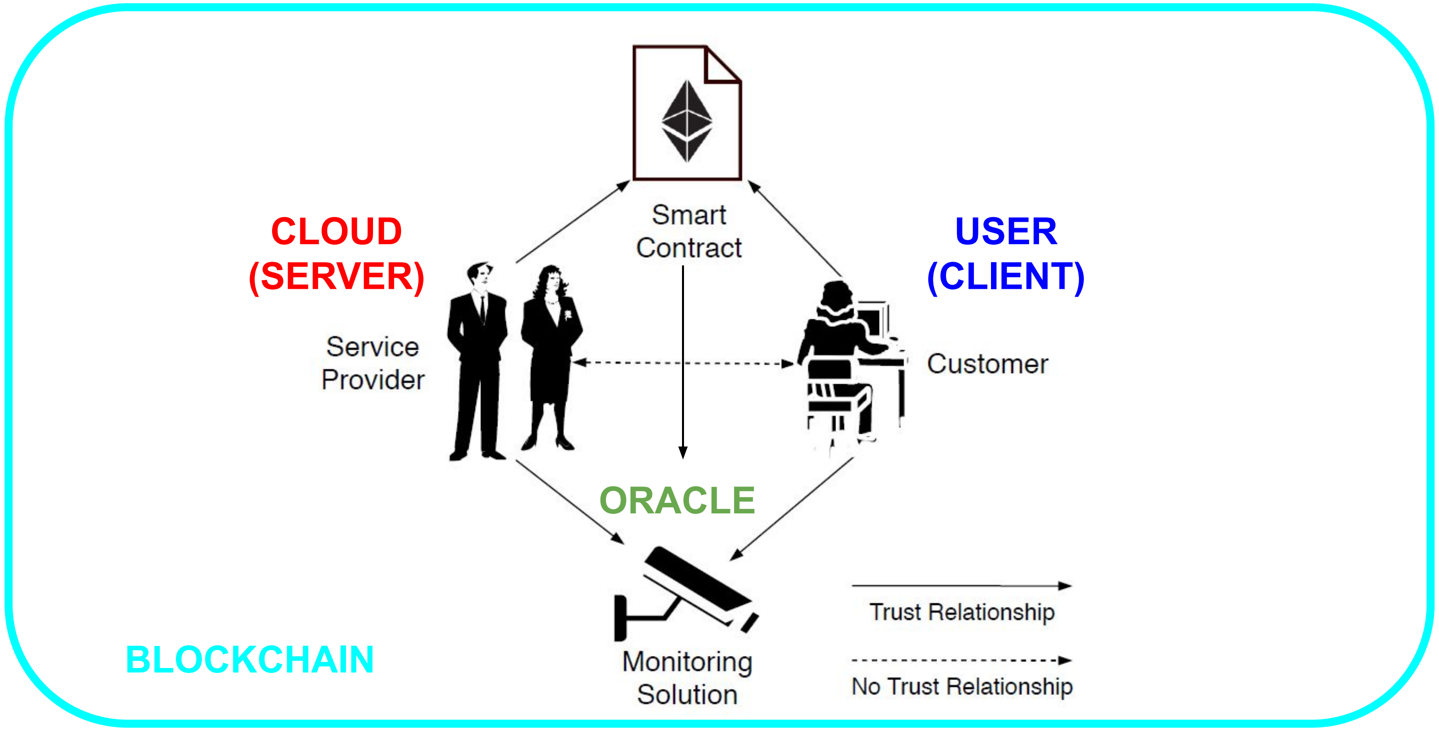
- ❑ URL content can be verified by an arbitrator
- ❖ Possible SLA violation
  - ➢ file is corrupted



- ❖ Possible SLA violation
  - ➢ file is not present but the user never asked for deletion

Implementation

- ❑ Architecture



- ❑ Framework → Truffle v.5.4.6

Implementation: Blockchain

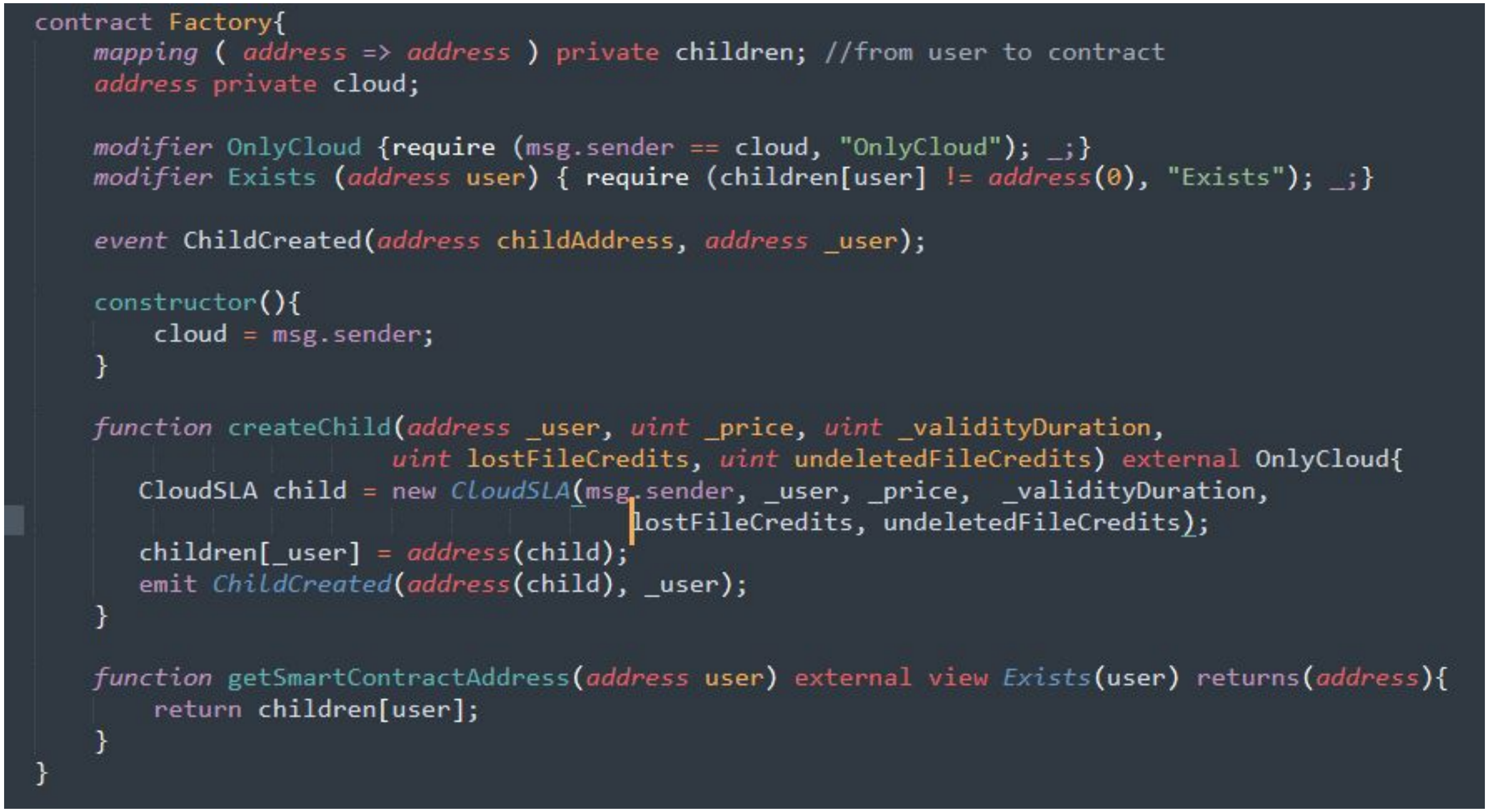
- **Permissioned** blockchain
- Proof-of-Authority consensus schemes
- **ConsenSys Quorum**
  - enterprise blockchain platform built on Quorum
  - Docker support
  - DLT
    a. Hyperledger Besu → IBFT2.0, QBFT
    b. GoQuorum → IBFT, QBFT, RAFT

- 8 nodes
  - 3 lightweight nodes
  - 4 validators
  - 1 RPC node

- HTTP + WebSockets

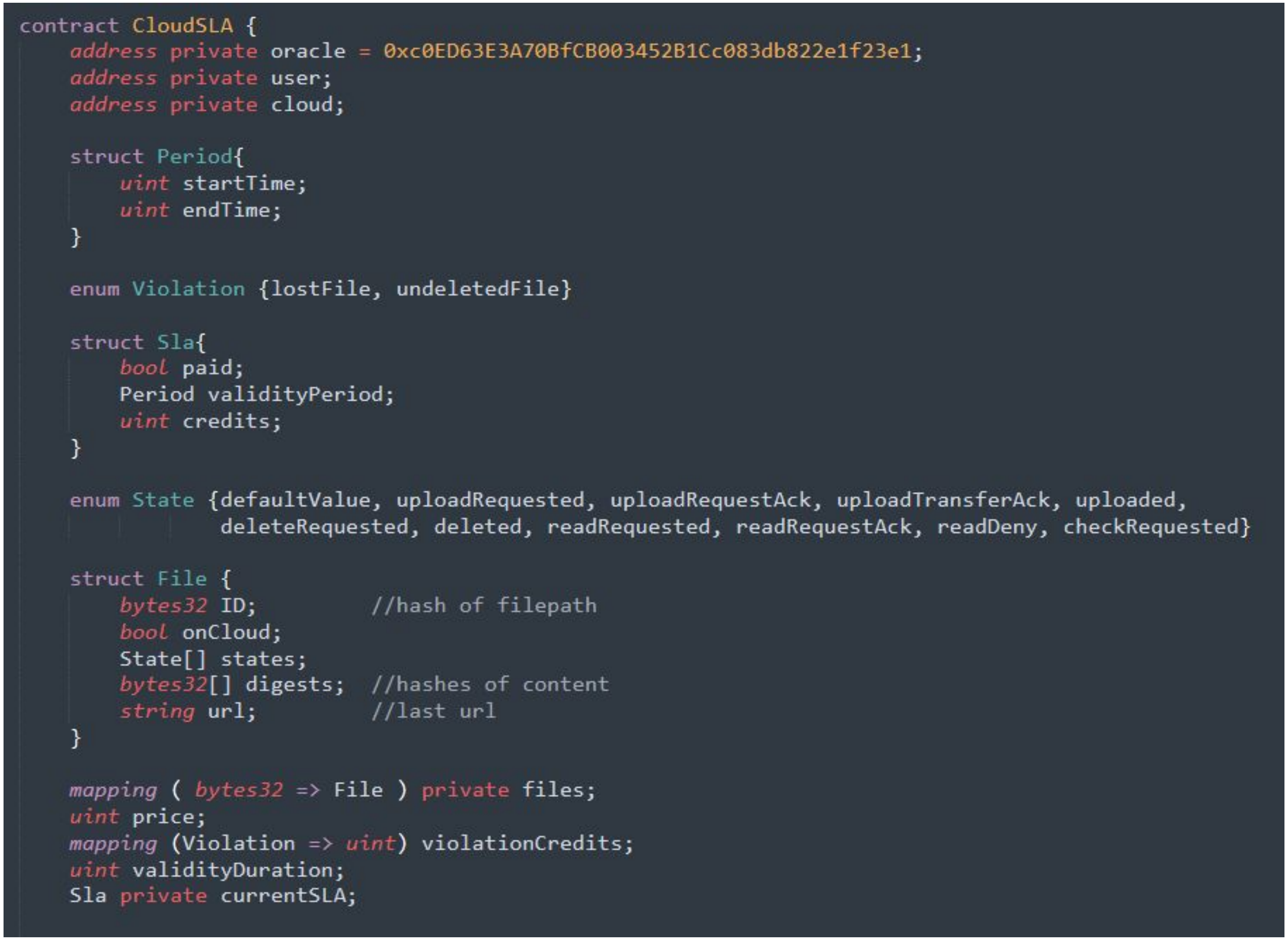
Implementation: Smart Contract

- Solidity v.0.8.0
- Interface to interact with Oracle contract
- Multiple users → *Factory* design pattern



```
contract Factory{
    mapping ( address => address ) private children; //from user to contract
    address private cloud;

    modifier OnlyCloud {require (msg.sender == cloud, "OnlyCloud"); _;}
    modifier Exists (address user) { require (children[user] != address(0), "Exists"); _;}

    event ChildCreated(address childAddress, address _user);

    constructor(){
        cloud = msg.sender;
    }

    function createChild(address _user, uint _price, uint _validityDuration,
                        uint lostFileCredits, uint undeletedFileCredits) external OnlyCloud{
       CloudSLA child = new CloudSLA(msg.sender, _user, _price,  _validityDuration,
                                    lostFileCredits, undeletedFileCredits);
       children[_user] = address(child);
       emit ChildCreated(address(child), _user);
    }

    function getSmartContractAddress(address user) external view Exists(user) returns(address){
        return children[user];
    }
}
```

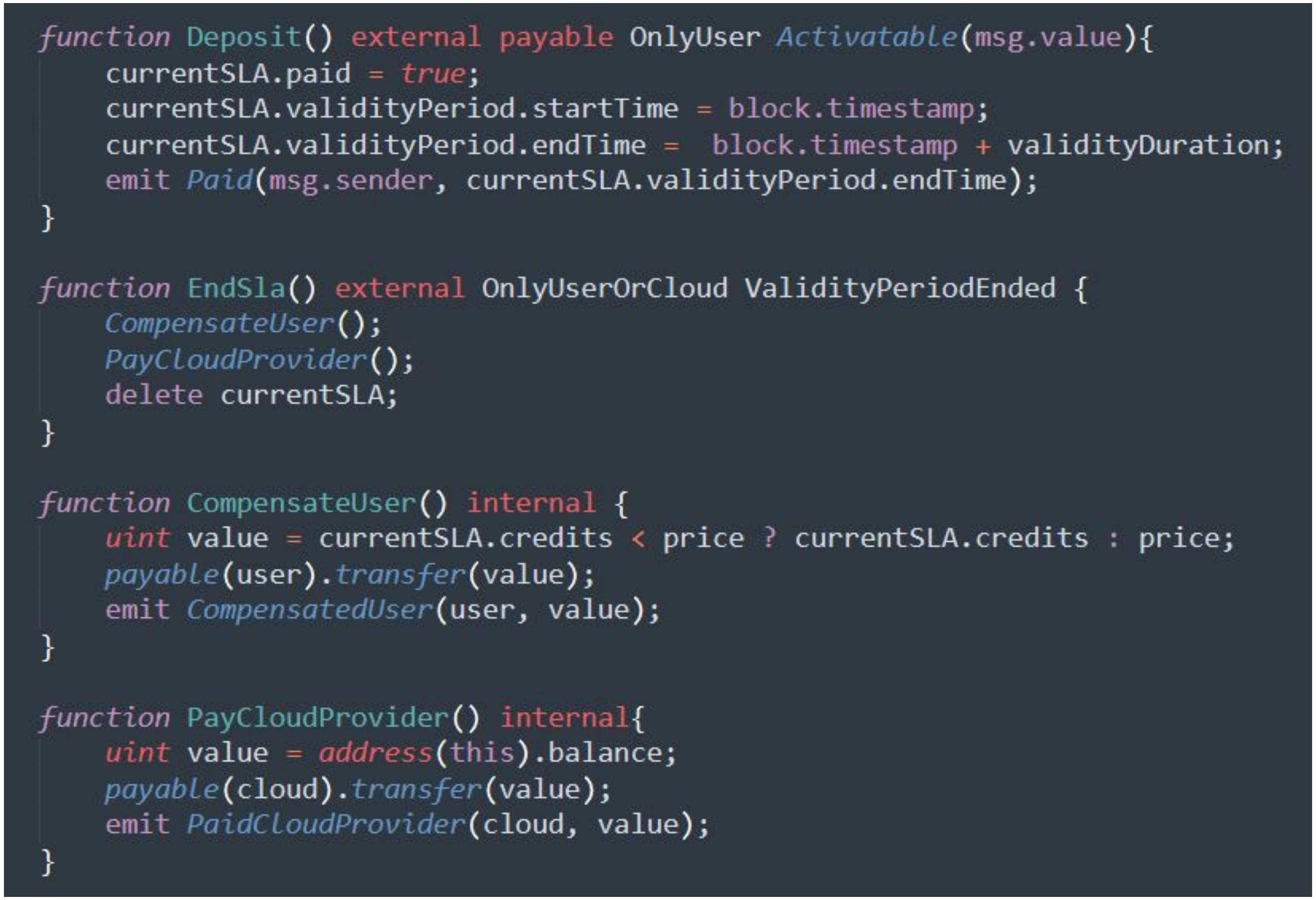
Implementation: Smart Contract



```
contract CloudSLA {
    address private oracle = 0xc0ED63E3A70BfCB003452B1Cc083db822e1f23e1;
    address private user;
    address private cloud;

    struct Period{
        uint startTime;
        uint endTime;
    }

    enum Violation {lostFile, undeletedFile}

    struct Sla{
        bool paid;
        Period validityPeriod;
        uint credits;
    }

    enum State {defaultValue, uploadRequested, uploadRequestAck, uploadTransferAck, uploaded,
                deleteRequested, deleted, readRequested, readRequestAck, readDeny, checkRequested}

    struct File {
        bytes32 ID;          //hash of filepath
        bool onCloud;
        State[] states;
        bytes32[] digests;   //hashes of content
        string url;          //last url
    }

    mapping ( bytes32 => File ) private files;
    uint price;
    mapping (Violation => uint) violationCredits;
    uint validityDuration;
    Sla private currentSLA;
```

Implementation: Smart Contract

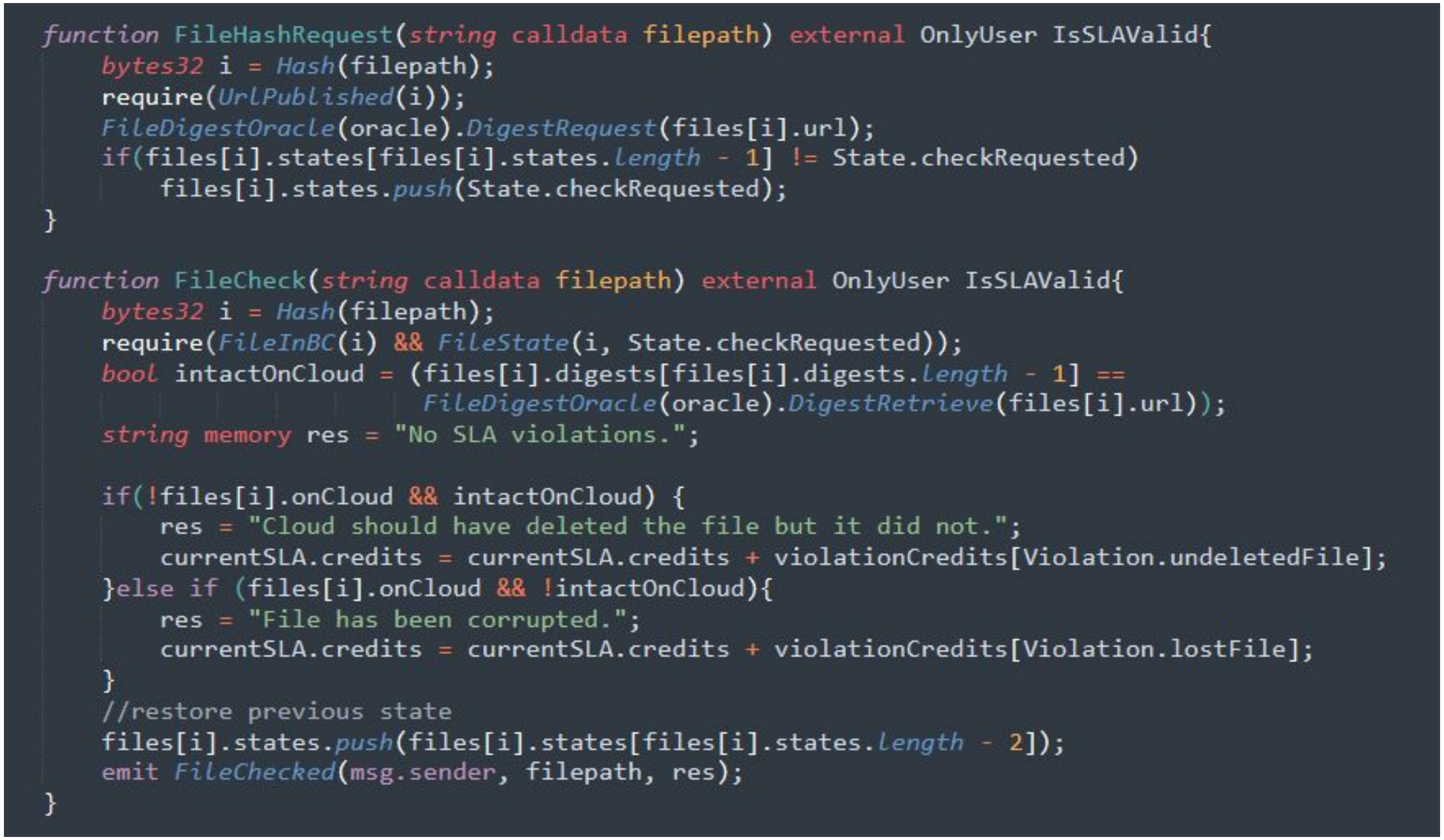
- Payment



```
function Deposit() external payable OnlyUser Activatable(msg.value){
    currentSLA.paid = true;
    currentSLA.validityPeriod.startTime = block.timestamp;
    currentSLA.validityPeriod.endTime =  block.timestamp + validityDuration;
    emit Paid(msg.sender, currentSLA.validityPeriod.endTime);
}

function EndSla() external OnlyUserOrCloud ValidityPeriodEnded {
    CompensateUser();
    PayCloudProvider();
    delete currentSLA;
}

function CompensateUser() internal {
    uint value = currentSLA.credits < price ? currentSLA.credits : price;
    payable(user).transfer(value);
    emit CompensatedUser(user, value);
}

function PayCloudProvider() internal{
    uint value = address(this).balance;
    payable(cloud).transfer(value);
    emit PaidCloudProvider(cloud, value);
}
```

Implementation: Smart Contract

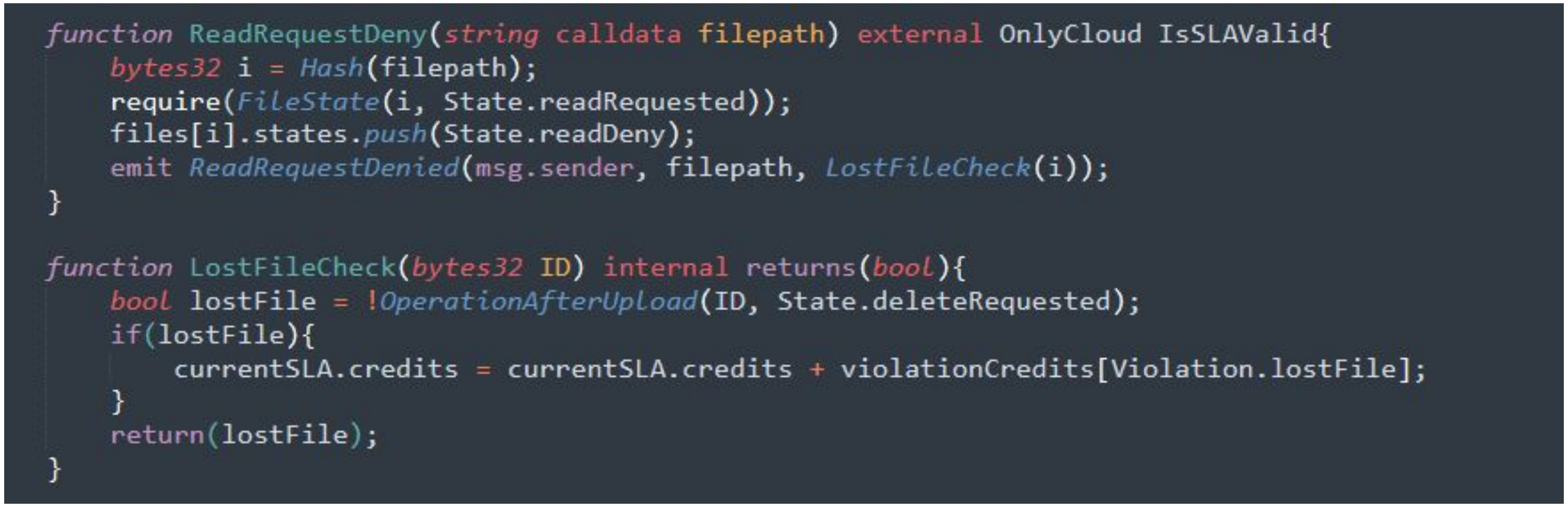
❑ SLA violation check



```
function FileHashRequest(string calldata filepath) external OnlyUser IsSLAValid{
    bytes32 i = Hash(filepath);
    require(UrlPublished(i));
    FileDigestOracle(oracle).DigestRequest(files[i].url);
    if(files[i].states[files[i].states.length - 1] != State.checkRequested)
        files[i].states.push(State.checkRequested);
}

function FileCheck(string calldata filepath) external OnlyUser IsSLAValid{
    bytes32 i = Hash(filepath);
    require(FileInBC(i) && FileState(i, State.checkRequested));
    bool intactOnCloud = (files[i].digests[files[i].digests.length - 1] ==
                          FileDigestOracle(oracle).DigestRetrieve(files[i].url));
    string memory res = "No SLA violations.";

    if(!files[i].onCloud && intactOnCloud) {
        res = "Cloud should have deleted the file but it did not.";
        currentSLA.credits = currentSLA.credits + violationCredits[Violation.undeletedFile];
    }else if (files[i].onCloud && !intactOnCloud){
        res = "File has been corrupted.";
        currentSLA.credits = currentSLA.credits + violationCredits[Violation.lostFile];
    }
    //restore previous state
    files[i].states.push(files[i].states[files[i].states.length - 2]);
    emit FileChecked(msg.sender, filepath, res);
}
```

Implementation: Smart Contract

- SLA violation check



```
function ReadRequestDeny(string calldata filepath) external OnlyCloud IsSLAValid{
    bytes32 i = Hash(filepath);
    require(FileState(i, State.readRequested));
    files[i].states.push(State.readDeny);
    emit ReadRequestDenied(msg.sender, filepath, LostFileCheck(i));
}

function LostFileCheck(bytes32 ID) internal returns(bool){
    bool lostFile = !OperationAfterUpload(ID, State.deleteRequested);
    if(lostFile){
        currentSLA.credits = currentSLA.credits + violationCredits[Violation.lostFile];
    }
    return(lostFile);
}
```

Implementation: Server

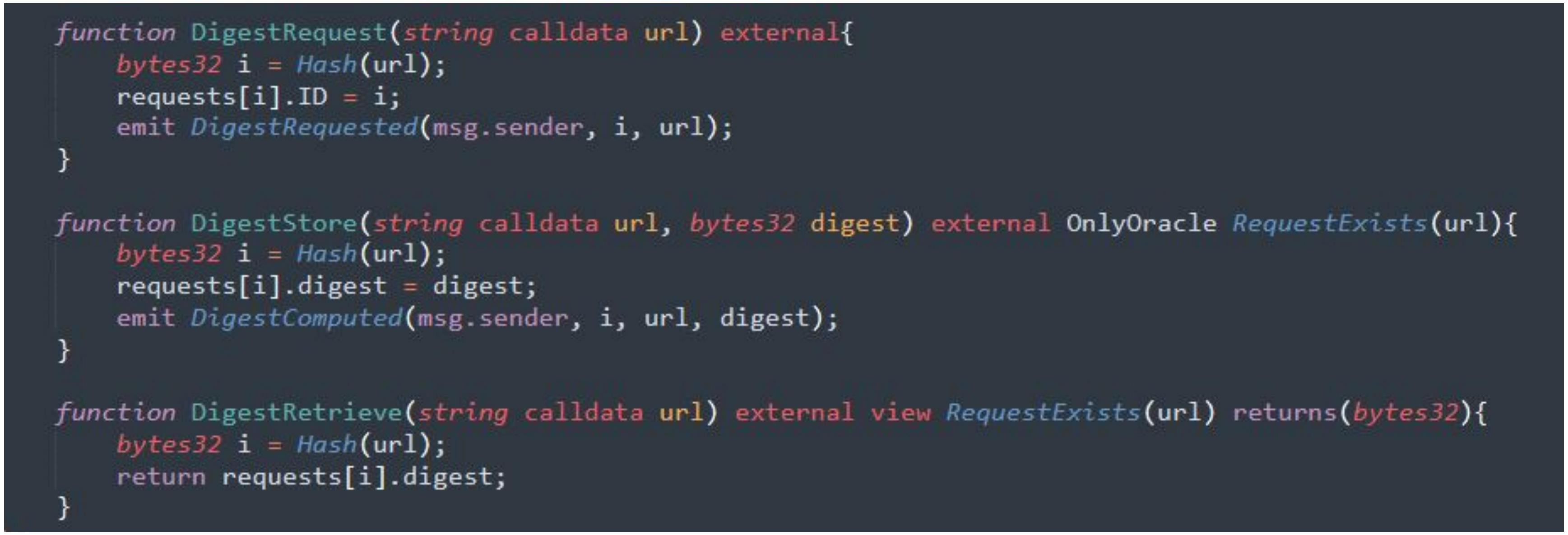
- **NodeJS Express**
- User authentication → *Auth0*
  - exception: blockchain-published url
- Smart contract interaction
  - transactions → *TruffleContract*
  - event subscriptions → *Web3*

  $\left.\right\rangle$ **WebsocketProvider**
- Contracts instances saved for later usage
- Scheduling of service termination → *node-schedule*

Implementation: Oracle

- **Server**
  - NodeJS Express
  - Web3 + TruffleContract
  - *Fetch API* to retrieve the file
  - *Crypto* module to compute SHA-256 hash

- **Smart Contract**



```
function DigestRequest(string calldata url) external{
    bytes32 i = Hash(url);
    requests[i].ID = i;
    emit DigestRequested(msg.sender, i, url);
}

function DigestStore(string calldata url, bytes32 digest) external OnlyOracle RequestExists(url){
    bytes32 i = Hash(url);
    requests[i].digest = digest;
    emit DigestComputed(msg.sender, i, url, digest);
}

function DigestRetrieve(string calldata url) external view RequestExists(url) returns(bytes32){
    bytes32 i = Hash(url);
    return requests[i].digest;
}
```

Implementation: Client

- Smart contract interaction
  - transactions ➞ *TruffleContract* — **Metamask** / **HttpProvider**
  - event subscriptions ➞ *Web3* — **WebsocketProvider**

- contract address
  - retrieved through *Factory* call
  - stored in *sessionStorage* for later usage

- AES file encryption and decryption
- SHA-256 hashing

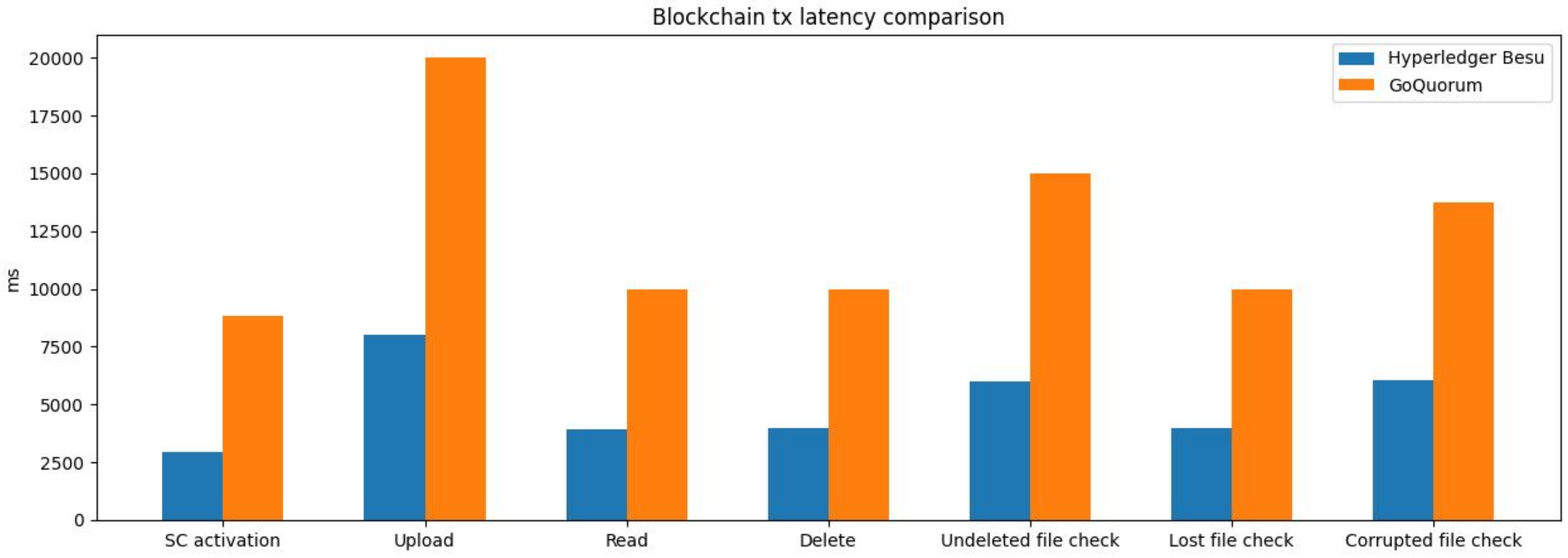
window.crypto

- Tx feedback + contract dashboard

Evaluation

Demo: https://cloudchain.com/mycloud/

Performance:



Conclusion

- The proposed blockchain-based cloud storage platform can provide an automatic settlement tool for SLA-related disputes

- Future work:
  - **Scalability** evaluation
  - Blockchain-based authentication
  - Smart contract negotiation
  - Transactions visualization

Code available at: https://github.com/emilypeek1/cloud-chain

References

[1] Gabriele D'Angelo, Stefano Ferretti, and Moreno Marzolla. 2018. A Blockchain-based Flight Data Recorder for Cloud Accountability. In Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems (CryBlock'18). Association for Computing Machinery, New York, NY, USA, 93–98.

[2] E. J. Scheid, B. B. Rodrigues, L. Z. Granville and B. Stiller. 2019. Enabling Dynamic SLA Compensation Using Blockchain-based Smart Contracts. IFIP/IEEE Symposium on Integrated Network and Service Management (IM), 2019, pp. 53-61.

[3] ConsenSys Quorum. Getting Started with ConsenSys Quorum.