CSE 1006

Blockchain And Cryptocurrency Technologies

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

DIGITAL ASSIGNMENT 2
SLOT A1 + TA1

SUNEHA GHOSH (18BCB0075)
SIDDHARTHA MONDAL (18BCB0145)

SUBMITTED TO:
PROF. BOOMINATHAN P.

Suneha

SOFTWARE REQUIREMENT

- Debian OS (we worked on Kali Linux)
- Nvm 9.11.1 version
- Npm
- The above two are required for running the Node Js files
- Yarn
- Memory : 20GB (for OS)

CODES

1. ClaimHolder.sol

```
pragma solidity ^0.4.22;

import './ERC735.sol';
import './KeyHolder.sol';

contract ClaimHolder is KeyHolder, ERC735 {

   mapping (bytes32 => Claim) claims;
   mapping (uint256 => bytes32[]) claimsByType;

   function addClaim(
      uint256 _claimType,
      uint256 _scheme,
      address _issuer,
      bytes _signature,
      bytes _data,
      string _uri
   )
      public
      returns (bytes32 claimRequestId)
   {
      bytes32 claimId = keccak256(_issuer, _claimType);

      if (msg.sender != address(this)) {
        require(keyHasPurpose(keccak256(msg.sender), 3), "Sender does not have claim signer key");
      }

      if (claims[claimId].issuer != _issuer) {
         claimsByType[_claimType].push(claimId);
      }

      claims[claimId].claimType = _claimType;
      claims[claimId].scheme = _scheme;
      claims[claimId].issuer = _issuer;
      claims[claimId].signature = _signature;
```

```
    claims[claimId].data = _data;
    claims[claimId].uri = _uri;

    emit ClaimAdded(
        claimId,
        _claimType,
        _scheme,
        _issuer,
        _signature,
        _data,
        _uri
    );

    return claimId;
}

function removeClaim(bytes32 _claimId) public returns (bool success) {
    if (msg.sender != address(this)) {
      require(keyHasPurpose(keccak256(msg.sender), 1), "Sender does not have management key");
    }

    /* uint index; */
    /* (index, ) = claimsByType[claims[_claimId].claimType].indexOf(_claimId);
    claimsByType[claims[_claimId].claimType].removeByIndex(index); */

    emit ClaimRemoved(
        _claimId,
        claims[_claimId].claimType,
        claims[_claimId].scheme,
        claims[_claimId].issuer,
        claims[_claimId].signature,
        claims[_claimId].data,
        claims[_claimId].uri
    );

    delete claims[_claimId];
    return true;
}

function getClaim(bytes32 _claimId)
    public
    constant
    returns(
        uint256 claimType,
        uint256 scheme,
        address issuer,
        bytes signature,
        bytes data,
        string uri
    )
```

```
  {
    return (
      claims[_claimId].claimType,
      claims[_claimId].scheme,
      claims[_claimId].issuer,
      claims[_claimId].signature,
      claims[_claimId].data,
      claims[_claimId].uri
    );
  }

  function getClaimIdsByType(uint256 _claimType)
    public
    constant
    returns(bytes32[] claimIds)
  {
    return claimsByType[_claimType];
  }

}
```

2. ClaimVerifier.sol

```
import './ClaimHolder.sol';

contract ClaimVerifier {

 event ClaimValid(ClaimHolder _identity, uint256 claimType);
 event ClaimInvalid(ClaimHolder _identity, uint256 claimType);

 ClaimHolder public trustedClaimHolder;

 function ClaimVerifier(address _trustedClaimHolder) public {
  trustedClaimHolder = ClaimHolder(_trustedClaimHolder);
 }

 function checkClaim(ClaimHolder _identity, uint256 claimType)
  public
  returns (bool claimValid)
 {
  if (claimIsValid(_identity, claimType)) {
   emit ClaimValid(_identity, claimType);
   return true;
  } else {
   emit ClaimInvalid(_identity, claimType);
   return false;
  }
 }

 function claimIsValid(ClaimHolder _identity, uint256 claimType)
```

```
  public
  constant
  returns (bool claimValid)
{
  uint256 foundClaimType;
  uint256 scheme;
  address issuer;
  bytes memory sig;
  bytes memory data;

  // Construct claimId (identifier + claim type)
  bytes32 claimId = keccak256(trustedClaimHolder, claimType);

  // Fetch claim from user
  ( foundClaimType, scheme, issuer, sig, data, ) = _identity.getClaim(claimId);

  bytes32 dataHash = keccak256(_identity, claimType, data);
  bytes32 prefixedHash = keccak256("\x19Ethereum Signed Message:\n32", dataHash);

  // Recover address of data signer
  address recovered = getRecoveredAddress(sig, prefixedHash);

  // Take hash of recovered address
  bytes32 hashedAddr = keccak256(recovered);

  // Does the trusted identifier have they key which signed the user's claim?
  return trustedClaimHolder.keyHasPurpose(hashedAddr, 3);
}

function getRecoveredAddress(bytes sig, bytes32 dataHash)
    public
    view
    returns (address addr)
{
    bytes32 ra;
    bytes32 sa;
    uint8 va;

    // Check the signature length
    if (sig.length != 65) {
      return (0);
    }

    // Divide the signature in r, s and v variables
    assembly {
      ra := mload(add(sig, 32))
      sa := mload(add(sig, 64))
      va := byte(0, mload(add(sig, 96)))
    }
```

```
    if (va < 27) {
      va += 27;
    }

    address recoveredAddress = ecrecover(dataHash, va, ra, sa);

    return (recoveredAddress);
  }

}
```

3. ERC725.sol

```
pragma solidity ^0.4.22;

contract ERC725 {

   uint256 constant MANAGEMENT_KEY = 1;
   uint256 constant ACTION_KEY = 2;
   uint256 constant CLAIM_SIGNER_KEY = 3;
   uint256 constant ENCRYPTION_KEY = 4;

   event KeyAdded(bytes32 indexed key, uint256 indexed purpose, uint256 indexed keyType);
   event KeyRemoved(bytes32 indexed key, uint256 indexed purpose, uint256 indexed keyType);
   event ExecutionRequested(uint256 indexed executionId, address indexed to, uint256 indexed value, bytes
data);
   event Executed(uint256 indexed executionId, address indexed to, uint256 indexed value, bytes data);
   event Approved(uint256 indexed executionId, bool approved);

   struct Key {
      uint256 purpose; //e.g., MANAGEMENT_KEY = 1, ACTION_KEY = 2, etc.
      uint256 keyType; // e.g. 1 = ECDSA, 2 = RSA, etc.
      bytes32 key;
   }

   function getKey(bytes32 _key) public constant returns(uint256 purpose, uint256 keyType, bytes32 key);
   function getKeyPurpose(bytes32 _key) public constant returns(uint256 purpose);
   function getKeysByPurpose(uint256 _purpose) public constant returns(bytes32[] keys);
   function addKey(bytes32 _key, uint256 _purpose, uint256 _keyType) public returns (bool success);
   function execute(address _to, uint256 _value, bytes _data) public returns (uint256 executionId);
   function approve(uint256 _id, bool _approve) public returns (bool success);
}
```

4. ERC735.sol

```
pragma solidity ^0.4.22;

contract ERC735 {
```

```
  event ClaimRequested(uint256 indexed claimRequestId, uint256 indexed claimType, uint256 scheme, address indexed issuer, bytes signature, bytes data, string uri);  event ClaimAdded(bytes32 indexed claimId, uint256 indexed claimType, address indexed issuer, uint256 signatureType, bytes32 signature, bytes claim, string uri);
  event ClaimAdded(bytes32 indexed claimId, uint256 indexed claimType, uint256 scheme, address indexed issuer, bytes signature, bytes data, string uri);
  event ClaimRemoved(bytes32 indexed claimId, uint256 indexed claimType, uint256 scheme, address indexed issuer, bytes signature, bytes data, string uri);
  event ClaimChanged(bytes32 indexed claimId, uint256 indexed claimType, uint256 scheme, address indexed issuer, bytes signature, bytes data, string uri);

  struct Claim {
    uint256 claimType;
    uint256 scheme;
    address issuer; // msg.sender
    bytes signature; // this.address + claimType + data
    bytes data;
    string uri;
  }

  function getClaim(bytes32 _claimId) public constant returns(uint256 claimType, uint256 scheme, address issuer, bytes signature, bytes data, string uri);
  function getClaimIdsByType(uint256 _claimType) public constant returns(bytes32[] claimIds);
  function addClaim(uint256 _claimType, uint256 _scheme, address issuer, bytes _signature, bytes _data, string _uri) public returns (bytes32 claimRequestId);
  function removeClaim(bytes32 _claimId) public returns (bool success);
}
```

5. Identity.sol

```
pragma solidity ^0.4.22;

import './ClaimHolder.sol';

contract Identity is ClaimHolder {

  function Identity(
    uint256[] _claimType,
    uint256[] _scheme,
    address[] _issuer,
    bytes _signature,
    bytes _data,
    string _uri,
    uint256[] _sigSizes,
    uint256[] dataSizes,
    uint256[] uriSizes
  )
    public
  {
    bytes32 claimId;
```

```
    uint offset = 0;
    uint uoffset = 0;
    uint doffset = 0;

    for (uint i = 0; i < _claimType.length; i++) {

        claimId = keccak256(_issuer[i], _claimType[i]);

        claims[claimId] = Claim(
            _claimType[i],
            _scheme[i],
            _issuer[i],
            getBytes(_signature, offset, _sigSizes[i]),
            getBytes(_data, doffset, dataSizes[i]),
            getString(_uri, uoffset, uriSizes[i])
        );

        offset += _sigSizes[i];
        uoffset += uriSizes[i];
        doffset += dataSizes[i];

        emit ClaimAdded(
            claimId,
            claims[claimId].claimType,
            claims[claimId].scheme,
            claims[claimId].issuer,
            claims[claimId].signature,
            claims[claimId].data,
            claims[claimId].uri
        );
    }
}

function getBytes(bytes _str, uint256 _offset, uint256 _length) constant returns (bytes) {
    bytes memory sig = new bytes(_length);
    uint256 j = 0;
    for (uint256 k = _offset; k< _offset + _length; k++) {
      sig[j] = _str[k];
      j++;
    }
    return sig;
}

function getString(string _str, uint256 _offset, uint256 _length) constant returns (string) {
    bytes memory strBytes = bytes(_str);
    bytes memory sig = new bytes(_length);
    uint256 j = 0;
    for (uint256 k = _offset; k< _offset + _length; k++) {
      sig[j] = strBytes[k];
      j++;
```

```
        }
        return string(sig);
    }
}
```

6. KeyHolder.sol

```
pragma solidity ^0.4.22;

import './ERC725.sol';

contract KeyHolder is ERC725 {

    uint256 executionNonce;

    struct Execution {
        address to;
        uint256 value;
        bytes data;
        bool approved;
        bool executed;
    }

    mapping (bytes32 => Key) keys;
    mapping (uint256 => bytes32[]) keysByPurpose;
    mapping (uint256 => Execution) executions;

    event ExecutionFailed(uint256 indexed executionId, address indexed to, uint256 indexed value, bytes data);

    function KeyHolder() public {
        bytes32 _key = keccak256(msg.sender);
        keys[_key].key = _key;
        keys[_key].purpose = 1;
        keys[_key].keyType = 1;
        keysByPurpose[1].push(_key);
        emit KeyAdded(_key, keys[_key].purpose, 1);
    }

    function getKey(bytes32 _key)
        public
        view
        returns(uint256 purpose, uint256 keyType, bytes32 key)
    {
        return (keys[_key].purpose, keys[_key].keyType, keys[_key].key);
    }

    function getKeyPurpose(bytes32 _key)
        public
        view
```

```
    returns(uint256 purpose)
  {
    return (keys[_key].purpose);
  }

  function getKeysByPurpose(uint256 _purpose)
    public
    view
    returns(bytes32[] _keys)
  {
    return keysByPurpose[_purpose];
  }

  function addKey(bytes32 _key, uint256 _purpose, uint256 _type)
    public
    returns (bool success)
  {
    require(keys[_key].key != _key, "Key already exists"); // Key should not already exist
    if (msg.sender != address(this)) {
      require(keyHasPurpose(keccak256(msg.sender), 1), "Sender does not have management key"); // Sender has MANAGEMENT_KEY
    }

    keys[_key].key = _key;
    keys[_key].purpose = _purpose;
    keys[_key].keyType = _type;

    keysByPurpose[_purpose].push(_key);

    emit KeyAdded(_key, _purpose, _type);

    return true;
  }

  function approve(uint256 _id, bool _approve)
    public
    returns (bool success)
  {
    require(keyHasPurpose(keccak256(msg.sender), 2), "Sender does not have action key");

    emit Approved(_id, _approve);

    if (_approve == true) {
      executions[_id].approved = true;
      success = executions[_id].to.call(executions[_id].data, 0);
      if (success) {
        executions[_id].executed = true;
        emit Executed(
          _id,
          executions[_id].to,
```

```
            executions[_id].value,
            executions[_id].data
        );
        return;
      } else {
        emit ExecutionFailed(
            _id,
            executions[_id].to,
            executions[_id].value,
            executions[_id].data
        );
        return;
      }
    } else {
      executions[_id].approved = false;
    }
    return true;
}

function execute(address _to, uint256 _value, bytes _data)
    public
    returns (uint256 executionId)
{
    require(!executions[executionNonce].executed, "Already executed");
    executions[executionNonce].to = _to;
    executions[executionNonce].value = _value;
    executions[executionNonce].data = _data;

    emit ExecutionRequested(executionNonce, _to, _value, _data);

    if (keyHasPurpose(keccak256(msg.sender),1) || keyHasPurpose(keccak256(msg.sender),2)) {
      approve(executionNonce, true);
    }

    executionNonce++;
    return executionNonce-1;
}

function removeKey(bytes32 _key)
    public
    returns (bool success)
{
    require(keys[_key].key == _key, "No such key");
    emit KeyRemoved(keys[_key].key, keys[_key].purpose, keys[_key].keyType);

    /* uint index;
    (index,) = keysByPurpose[keys[_key].purpose.indexOf(_key);
    keysByPurpose[keys[_key].purpose.removeByIndex(index); */

    delete keys[_key];
```

```
        return true;
    }

    function keyHasPurpose(bytes32 _key, uint256 _purpose)
        public
        view
        returns(bool result)
    {
        bool isThere;
        if (keys[_key].key == 0) return false;
        isThere = keys[_key].purpose <= _purpose;
        return isThere;
    }

}
```

7. _github.js

```
var OAuth = require('oauth').OAuth2
var HTML = require('./html')
var superagent = require('superagent')

const ClaimType = 6 // Has Google

module.exports = function google(app, { web3, googleApp, baseUrl }) {
  const redirect_uri = `${baseUrl}/google-auth-response`

  var googleOAuth = new OAuth(
    googleApp.client_id,
    googleApp.secret,
    'https://accounts.google.com',
    '/o/oauth2/auth',
    '/o/oauth2/token'
  )

  app.get('/google-auth', (req, res) => {
    if (!req.query.target) {
      res.send('No target identity contract provided')
      return
    }
    if (!req.query.issuer) {
      res.send('No issuer identity contract provided')
      return
    }

    req.session.targetIdentity = req.query.target
    req.session.issuer = req.query.issuer
    req.session.state = web3.utils.randomHex(8)
```

```
  var authURL = googleOAuth.getAuthorizeUrl({
    redirect_uri,
    scope: 'https://www.googleapis.com/auth/userinfo.profile',
    state: req.session.state,
    response_type: 'code'
  })

  res.redirect(authURL)
})

app.get(
  '/google-auth-response',
  (req, res, next) => {
    googleOAuth.getOAuthAccessToken(
      req.query.code,
      {
        redirect_uri,
        grant_type: 'authorization_code'
      },
      function(e, access_token, refresh_token, results) {
        if (e) {
          next(e)
        } else if (results.error) {
          next(results.error)
        } else {
          req.access_token = access_token
          next()
        }
      }
    )
  },
  (req, res, next) => {
    superagent
      .get('https://www.googleapis.com/oauth2/v1/userinfo')
      .query({
        alt: 'json',
        access_token: req.access_token
      })
      .then(response => {
        req.googleUser = response.body
        next()
      })
  },
  async (req, res) => {
    // var data = JSON.stringify({ user_id: req.googleUser.id })

    var rawData = 'Verified OK'
    var hexData = web3.utils.asciiToHex(rawData)
    var hashed = web3.utils.soliditySha3(req.session.targetIdentity, ClaimType, hexData)
```

```
    req.signedData = await web3.eth.accounts.sign(hashed, googleApp.claimSignerKey)

    res.send(
      HTML(`
      <div class="mb-2">Successfully signed claim:</div>
      <div class="mb-2"><b>Issuer:</b> ${req.session.issuer}</div>
      <div class="mb-2"><b>Target:</b> ${req.session.targetIdentity}</div>
      <div class="mb-2"><b>Data:</b> ${rawData}</div>
      <div class="mb-2"><b>Signature:</b> ${req.signedData.signature}</div>
      <div class="mb-2"><b>Hash:</b> ${req.signedData.messageHash}</div>
      <div><button class="btn btn-primary" onclick="window.done()">OK</button></div>
      <script>
        window.done = function() {
          window.opener.postMessage('signed-data:${
            req.signedData.signature
          }:${rawData}:${ClaimType}', '*')
        }
      </script>`)
    )
  }
 )
}
```

8. _google.js

```
var OAuth = require('oauth').OAuth2
var HTML = require('./html')
var superagent = require('superagent')

const ClaimType = 6 // Has Google

module.exports = function google(app, { web3, googleApp, baseUrl }) {
 const redirect_uri = `${baseUrl}/google-auth-response`

 var googleOAuth = new OAuth(
  googleApp.client_id,
  googleApp.secret,
  'https://accounts.google.com',
  '/o/google/auth',
  '/o/google/token'
 )

 app.get('/google-auth', (req, res) => {
  if (!req.query.target) {
   res.send('No target identity contract provided')
   return
  }
  if (!req.query.issuer) {
   res.send('No issuer identity contract provided')
   return
  }
```

```
  req.session.targetIdentity = req.query.target
  req.session.issuer = req.query.issuer
  req.session.state = web3.utils.randomHex(8)

  var authURL = googleOAuth.getAuthorizeUrl({
    redirect_uri,
    scope: 'https://www.googleapis.com/auth/userinfo.profile',
    state: req.session.state,
    response_type: 'code'
  })

  res.redirect(authURL)
})

app.get(
  '/google-auth-response',
  (req, res, next) => {
    googleOAuth.getOAuthAccessToken(
      req.query.code,
      {
        redirect_uri,
        grant_type: 'authorization_code'
      },
      function(e, access_token, refresh_token, results) {
        if (e) {
          next(e)
        } else if (results.error) {
          next(results.error)
        } else {
          req.access_token = access_token
          next()
        }
      }
    )
  },
  (req, res, next) => {
    superagent
      .get('https://www.googleapis.com/google/v1/userinfo')
      .query({
        alt: 'json',
        access_token: req.access_token
      })
      .then(response => {
        req.googleUser = response.body
        next()
      })
  },
  async (req, res) => {
    // var data = JSON.stringify({ user_id: req.googleUser.id })
```

```
    var rawData = 'Verified OK'
    var hexData = web3.utils.asciiToHex(rawData)
    var hashed = web3.utils.solioditySha3(req.session.targetIdentity, ClaimType, hexData)
    req.signedData = await web3.eth.accounts.sign(hashed, googleApp.claimSignerKey)

    res.send(
      HTML(`
      <div class="mb-2">Successfully signed claim:</div>
      <div class="mb-2"><b>Issuer:</b> ${req.session.issuer}</div>
      <div class="mb-2"><b>Target:</b> ${req.session.targetIdentity}</div>
      <div class="mb-2"><b>Data:</b> ${rawData}</div>
      <div class="mb-2"><b>Signature:</b> ${req.signedData.signature}</div>
      <div class="mb-2"><b>Hash:</b> ${req.signedData.messageHash}</div>
      <div><button class="btn btn-primary" onclick="window.done()">OK</button></div>
      <script>
        window.done = function() {
          window.opener.postMessage('signed-data:${
            req.signedData.signature
          }:${rawData}:${ClaimType}', '*')
        }
      </script>`)
    )
  }
 )
}
```

9. accounts.js

```
var bip39 = require('bip39')
var HDKey = require('hdkey')
var Web3 = require('web3')
var web3 = new Web3()

const mnemonic = process.argv.slice(2).join(' ')

if (!mnemonic) {
  console.log("\nUsage: node accounts.js [mnemonic]")
  console.log("eg node accounts.js candy maple cake sugar pudding cream honey rich smooth crumble sweet
treat\n")
  process.exit()
}

console.log(`\nMnemonic: ${mnemonic}\n`)

for (var offset = 0; offset < 10; offset++) {
  var seed = bip39.mnemonicToSeed(mnemonic)
  var acct = HDKey.fromMasterSeed(seed).derive("m/44'/60'/0'/0/" + offset)
  var account = web3.eth.accounts.privateKeyToAccount(
    `0x${acct.privateKey.toString('hex')}`
```

```
  )
  console.log(`${account.address} - ${account.privateKey}`)
}

console.log()
process.exit()
```

10. Deploy.js

```
import helper from '../test/_helper'

;(async () => {
  var { accounts, web3, deploy, server } = await helper(
    `${__dirname}/identity/`,
    'https://rinkeby.infura.io'
    // 'http://localhost:8545'
  )

  var account = web3.eth.accounts.wallet.add(
    '0xPRIV_KEY'
  )

  await deploy('ClaimHolder', { from: account, log: true })

  if (server) {
    server.close()
  }
})()
```

11. Identity.js

```
import keyMirror from 'utils/keyMirror'
import Identity from '../contracts/Identity'
import ClaimHolder from '../contracts/ClaimHolder'
import ClaimVerifier from '../contracts/ClaimVerifier'

import { updateBalance } from './Wallet'

if (typeof window !== 'undefined') {
  window.contracts = {
    ClaimHolder: addr => new web3.eth.Contract(ClaimHolder.abi, addr),
    ClaimVerifier: addr => new web3.eth.Contract(ClaimVerifier.abi, addr),
    Identity: addr => {
      var i = new web3.eth.Contract(Identity.abi, addr)
      i.options.data = '0x' + Identity.data
      return i
    }
  }
}

function lookup(Types) {
```

```
  return function claimType(id) {
    var type = Types.find(t => t.id === id)
    return type ? type.value : id
  }
}

const chainConstants = c => ({
  [`${c}`]: null,
  [`${c}_HASH`]: null,
  [`${c}_RECEIPT`]: null,
  [`${c}_SUCCESS`]: null,
  [`${c}_CONFIRMATION`]: null,
  [`${c}_ERROR`]: null
})

export const KeyPurpose = [
  { id: '1', value: 'Management' },
  { id: '2', value: 'Action' },
  { id: '3', value: 'Claim Signer' },
  { id: '4', value: 'Encryption' }
]

export function keyPurpose(id) {
  var keyPurpose = KeyPurpose.find(t => t.id === id)
  return keyPurpose ? keyPurpose.value : id
}

export const KeyTypes = [{ id: '1', value: 'ECDSA' }, { id: '2', value: 'RSA' }]
export const keyType = lookup(KeyTypes)

export const Schemes = [
  { id: '1', value: 'ECDSA' },
  { id: '2', value: 'RSA' },
  { id: '3', value: 'Contract Call' },
  { id: '4', value: 'Self-Claim' }
]
export const scheme = lookup(Schemes)

export const ClaimTypes = [
  { id: '10', value: 'Full Name' },
  { id: '11', value: 'Blockchain Profile' },
  { id: '8', value: 'Email' },
  { id: '3', value: 'Has Facebook' },
  { id: '4', value: 'Has Twitter' },
  { id: '5', value: 'Has GitHub' },
  { id: '6', value: 'Has Google' },
  { id: '9', value: 'Has LinkedIn' },
  { id: '7', value: 'Verified' }
]
export const claimType = lookup(ClaimTypes)
```

```
export const IdentityConstants = keyMirror(
  {
    FIND: null,
    FIND_SUCCESS: null,
    REMOVE: null,
    REMOVE_SUCCESS: null,
    REMOVE_VERIFIER: null,
    GET_EVENTS: null,
    GET_EVENTS_SUCCESS: null,
    RESET: null,
    IMPORT: null,
    ...chainConstants('DEPLOY'),
    ...chainConstants('DEPLOY_VERIFIER'),
    ...chainConstants('ADD_KEY'),
    ...chainConstants('REMOVE_KEY'),
    ...chainConstants('ADD_CLAIM'),
    ...chainConstants('REMOVE_CLAIM'),
    ...chainConstants('APPROVE_EXECUTION'),
    ...chainConstants('CHECK_CLAIM')
  },
  'IDENTITY'
)

export function deployIdentityContract(
  name,
  identityType,
  uri,
  preAdd,
  icon,
  signerServices
) {
  return async function(dispatch) {
    var RawContract = preAdd ? Identity : ClaimHolder

    var Contract = new web3.eth.Contract(RawContract.abi)
    var tx = Contract.deploy({
      data: '0x' + RawContract.data,
      arguments: preAdd
    }).send({ gas: 4612388, from: web3.eth.defaultAccount })

    var data = {
      name,
      identityType,
      uri,
      preAdd,
      icon,
      owner: web3.eth.defaultAccount,
      signerServices
    }
```

```
    dispatch(sendTransaction(tx, IdentityConstants.DEPLOY, data))
  }
}

export function importIdentityContract(address, name) {
  return async function(dispatch) {
    var Contract = new web3.eth.Contract(ClaimHolder.abi, address)
    var events = await Contract.getPastEvents('allEvents', { fromBlock: 0 })
    var tx = await web3.eth.getTransaction(events[0].transactionHash)

    dispatch({
      type: IdentityConstants.IMPORT,
      name,
      address,
      identityType: 'identity',
      owner: tx.from
    })
  }
}

export function deployClaimVerifier(args) {
  return async function(dispatch) {
    var Contract = new web3.eth.Contract(ClaimVerifier.abi)
    var tx = Contract.deploy({
      data: '0x' + ClaimVerifier.data,
      arguments: [args.trustedIdentity]
    }).send({ gas: 4612388, from: web3.eth.defaultAccount })

    var data = {
      ...args,
      owner: web3.eth.defaultAccount
    }

    dispatch(sendTransaction(tx, IdentityConstants.DEPLOY_VERIFIER, data))
  }
}

export function addKey({ purpose, keyType, key, identity }) {
  return async function(dispatch) {
    var data = { purpose, keyType, key, identity }

    const Contract = new web3.eth.Contract(ClaimHolder.abi, identity)

    try {
      Contract.methods.addKey(key, purpose, keyType).encodeABI()
    } catch (e) {
      dispatch({
        type: IdentityConstants.ADD_KEY_ERROR,
        error: e,
```

```
      message: e.message
    })
    return
  }

  var tx = Contract.methods.addKey(key, purpose, keyType).send({
    gas: 4612388,
    from: web3.eth.defaultAccount
  })

  dispatch(
    sendTransaction(tx, IdentityConstants.ADD_KEY, data, () => {
      dispatch(getEvents('ClaimHolder', identity))
    })
  )
 }
}

export function removeKey({ identity, key }) {
 return async function(dispatch) {
  const Contract = new web3.eth.Contract(ClaimHolder.abi, identity)
  var tx = Contract.methods.removeKey(key).send({
    from: web3.eth.defaultAccount,
    gas: 3000000
  })

  dispatch(
    sendTransaction(tx, IdentityConstants.REMOVE_KEY, null, () => {
      dispatch(getEvents('ClaimHolder', identity))
    })
  )
 }
}

export function approveExecution(identity, executionId) {
 return async function(dispatch) {
  const Contract = new web3.eth.Contract(ClaimHolder.abi, identity)
  var tx = Contract.methods.approve(executionId, true).send({
    from: web3.eth.defaultAccount,
    gas: 3000000
  })

  dispatch(
    sendTransaction(tx, IdentityConstants.APPROVE_EXECUTION, null, () => {
      dispatch(getEvents('ClaimHolder', identity))
    })
  )
 }
}
```

```
export function checkClaim(verifier, identity, claimType) {
  return function(dispatch) {
    dispatch({ type: IdentityConstants.CHECK_CLAIM })

    const Contract = new web3.eth.Contract(ClaimVerifier.abi, verifier)

    var tx = Contract.methods.checkClaim(identity, claimType).send({
      from: web3.eth.defaultAccount,
      gas: 3000000
    })

    dispatch(
      sendTransaction(tx, IdentityConstants.CHECK_CLAIM, {}, () => {
        dispatch(getEvents('ClaimVerifier', verifier))
      })
    )
  }
}

export function getEvents(type, address) {
  return async function(dispatch) {
    dispatch({ type: IdentityConstants.GET_EVENTS })

    var contract = new web3.eth.Contract(
      type === 'ClaimHolder' ? ClaimHolder.abi : ClaimVerifier.abi,
      address
    )

    var events = await contract.getPastEvents('allEvents', {
      fromBlock: 0,
      toBlock: 'latest'
    })

    dispatch({ type: IdentityConstants.GET_EVENTS_SUCCESS, events })
  }
}

export function removeIdentity(address) {
  return { type: IdentityConstants.REMOVE, address }
}

export function removeVerifier(address) {
  return { type: IdentityConstants.REMOVE_VERIFIER, address }
}

export function reset() {
  return { type: IdentityConstants.RESET }
}

export function addClaim({
```

```
  data,
  claimIssuer,
  targetIdentity,
  uri,
  claimType,
  scheme,
  signature,
  refresh
}) {
  return async function(dispatch) {
    var txData = {
      data,
      claimIssuer,
      targetIdentity,
      uri,
      claimType,
      scheme,
      signature
    }

    var hexedData = web3.utils.asciiToHex(data)

    if (!signature) {
      if (String(scheme) === '1') {
        signature = await web3.eth.sign(
          web3.utils.solidity Sha3(targetIdentity, claimType, hexedData),
          web3.eth.defaultAccount
        )
      } else {
        signature = web3.utils.asciiToHex('')
      }
    }

    var UserIdentity = new web3.eth.Contract(ClaimHolder.abi, targetIdentity)

    var abi = await UserIdentity.methods
      .addClaim(claimType, scheme, claimIssuer, signature, hexedData, uri)
      .encodeABI()

    var tx = UserIdentity.methods.execute(targetIdentity, 0, abi).send({
      gas: 4612388,
      from: web3.eth.defaultAccount
    })

    dispatch(
      sendTransaction(tx, IdentityConstants.ADD_CLAIM, txData, () => {
        if (refresh) {
          dispatch(getEvents('ClaimHolder', targetIdentity))
        }
      })
```

```
    )
  }
}

export function removeClaim({ identity, claim }) {
  return async function(dispatch, getState) {
    dispatch({ type: IdentityConstants.REMOVE_CLAIM, claim })

    var state = getState()

    const Contract = new web3.eth.Contract(ClaimHolder.abi, identity)

    var tx = Contract.methods.removeClaim(claim).send({
      from: state.wallet.activeAddress,
      gas: 3000000
    })

    dispatch(
      sendTransaction(tx, IdentityConstants.REMOVE_CLAIM, {}, () => {
        dispatch(getEvents('ClaimHolder', identity))
      })
    )
  }
}

function sendTransaction(transaction, type, data, callback) {
  return async function(dispatch) {
    dispatch({ type, ...data })

    transaction
      .on('error', error => {
        dispatch({
          type: `${type}_ERROR`,
          message: error.message
        })
      })
      .on('transactionHash', hash => {
        dispatch({ type: `${type}_HASH`, hash })
      })
      .on('receipt', receipt => {
        dispatch({ type: `${type}_RECEIPT`, receipt, ...data })
        dispatch(updateBalance())
      })
      .on('confirmation', num => {
        dispatch({ type: `${type}_CONFIRMATION`, num })
      })
      .then(receipt => {
        dispatch({ type: `${type}_SUCCESS`, receipt, ...data })
        if (callback) {
          callback()
```

```
      }
    })
    .catch(err => {
      dispatch({
        type: `${type}_ERROR`,
        message: err.message
      })
    })
  }
}
```

12. Wallet.js

```
import keyMirror from 'utils/keyMirror'
import balance from 'utils/balance'

export const WalletConstants = keyMirror(
  {
    LOAD: null,
    LOAD_SUCCESS: null,
    LOAD_ERROR: null,

    LOAD_EXTERNAL: null,
    LOAD_EXTERNAL_SUCCESS: null,
    LOAD_EXTERNAL_ERROR: null,

    ADD_ACCOUNT: null,
    ADD_ACCOUNT_SUCCESS: null,
    ADD_ACCOUNT_ERROR: null,

    SELECT_ACCOUNT: null,
    SELECT_ACCOUNT_SUCCESS: null,
    SELECT_ACCOUNT_ERROR: null,

    IMPORT_ACCOUNT: null,
    IMPORT_ACCOUNT_SUCCESS: null,
    IMPORT_ACCOUNT_ERROR: null,

    SAVE: null,
    SAVE_SUCCESS: null,
    SAVE_ERROR: null,

    REMOVE_ACCOUNT: null,
    REMOVE_ACCOUNT_SUCCESS: null,
    REMOVE_ACCOUNT_ERROR: null,

    UPDATE_BALANCE: null,
    UPDATE_BALANCE_SUCCESS: null,
    UPDATE_BALANCE_ERROR: null,
```

```
    SET_CURRENCY: null,
    LOCK_WALLET: null,
    UNLOCK_WALLET: null,
    UNLOCKED_WALLET: null
  },
  'WALLET'
)

var watchMetaMaskInterval
function watchMetaMask(dispatch, currentAccount) {
  watchMetaMaskInterval = setInterval(async function() {
    var accounts = await web3.eth.getAccounts()
    if (currentAccount !== accounts[0]) {
      dispatch(loadWallet(true))
    }
  }, 1000)
}

export function loadWallet(external) {
  return async function(dispatch, getState) {
    var state = getState()

    dispatch({ type: WalletConstants.LOAD, external })
    var wallet = web3.eth.accounts.wallet,
      accounts = [],
      balanceWei,
      balances = {}

    clearInterval(watchMetaMaskInterval)

    try {
      if (external) {
        web3.setProvider(state.network.browserProvider)
        accounts = await web3.eth.getAccounts()

        balanceWei = await web3.eth.getBalance(accounts[0])
        balances[accounts[0]] = balance(balanceWei, state.wallet.exchangeRates)

        web3.eth.accounts.wallet.clear()
        web3.eth.defaultAccount = accounts[0]
        dispatch({
          type: WalletConstants.LOAD_EXTERNAL_SUCCESS,
          activeAddress: accounts[0],
          balances
        })
        watchMetaMask(dispatch, accounts[0])
        return
      }

      web3.setProvider(state.network.provider)
```

```
    // wallet.load is expensive, so cache private keys in sessionStorage
    if (window.sessionStorage.privateKeys) {
      JSON.parse(window.sessionStorage.privateKeys).forEach(key =>
        web3.eth.accounts.wallet.add(key)
      )
    } else {
      wallet = web3.eth.accounts.wallet.load('', 'blockchain-identity')

      var accountKeys = []
      for (var k = 0; k < wallet.length; k++) {
        accountKeys.push(wallet[k].privateKey)
      }
      if (accountKeys.length) {
        window.sessionStorage.privateKeys = JSON.stringify(accountKeys)
      }
    }

    for (var i = 0; i < wallet.length; i++) {
      accounts.push(wallet[i].address)
    }

    for (let hash of accounts) {
      balanceWei = await web3.eth.getBalance(hash)
      balances[hash] = balance(balanceWei, state.wallet.exchangeRates)
    }

    web3.eth.defaultAccount = accounts[0]

    dispatch({
      type: WalletConstants.LOAD_SUCCESS,
      wallet,
      accounts,
      balances
    })
  } catch (error) {
    dispatch({ type: WalletConstants.LOAD_ERROR, error })
  }
 }
}

export function selectAccount(address) {
 return async function(dispatch) {
  dispatch({ type: WalletConstants.SELECT_ACCOUNT, address })

  try {
    var account = web3.eth.accounts.wallet[address]
    web3.eth.defaultAccount = address

    dispatch({
```

```
      type: WalletConstants.SELECT_ACCOUNT_SUCCESS,
      account,
      activeAddress: address
    })
  } catch (error) {
    dispatch({ type: WalletConstants.SELECT_ACCOUNT_ERROR, error })
  }
 }
}

export function addAccount() {
 return async function(dispatch) {
  dispatch({ type: WalletConstants.ADD_ACCOUNT })

  try {
    var wallet = web3.eth.accounts.wallet.create(1),
      account = wallet[wallet.length - 1]

    dispatch({
      type: WalletConstants.ADD_ACCOUNT_SUCCESS,
      wallet,
      account
    })
  } catch (error) {
    dispatch({ type: WalletConstants.ADD_ACCOUNT_ERROR, error })
  }
 }
}

export function importAccountFromKey(privateKey) {
 return async function(dispatch, getState) {
  var state = getState()
  dispatch({ type: WalletConstants.IMPORT_ACCOUNT })

  try {
    var account = web3.eth.accounts.wallet.add(privateKey)
    var wallet = web3.eth.accounts.wallet

    var balanceWei = await web3.eth.getBalance(account.address)

    dispatch({
      type: WalletConstants.IMPORT_ACCOUNT_SUCCESS,
      account: wallet[wallet.length - 1],
      wallet,
      balance: balance(balanceWei, state.wallet.exchangeRates)
    })
  } catch (error) {
    dispatch({ type: WalletConstants.IMPORT_ACCOUNT_ERROR, error })
  }
 }
```

```
}

export function removeAccount(hash) {
  return async function(dispatch) {
    dispatch({ type: WalletConstants.REMOVE_ACCOUNT })

    try {
      var wallet = web3.eth.accounts.wallet.remove(hash)

      dispatch({
        type: WalletConstants.REMOVE_ACCOUNT_SUCCESS,
        hash,
        wallet
      })
    } catch (error) {
      dispatch({ type: WalletConstants.REMOVE_ACCOUNT_ERROR, error })
    }
  }
}

export function saveWallet() {
  return async function(dispatch) {
    dispatch({ type: WalletConstants.SAVE })

    try {
      web3.eth.accounts.wallet.save('', 'blockchain-identity')
      dispatch({ type: WalletConstants.SAVE_SUCCESS })
    } catch (error) {
      dispatch({ type: WalletConstants.SAVE_ERROR, error })
    }
  }
}

export function updateBalance() {
  return async function(dispatch, getState) {
    dispatch({ type: WalletConstants.UPDATE_BALANCE })

    var state = getState()
    var account = state.wallet.activeAddress

    var wei = await web3.eth.getBalance(account)

    dispatch({
      type: WalletConstants.UPDATE_BALANCE_SUCCESS,
      account,
      balance: balance(wei, state.wallet.exchangeRates)
    })
  }
}
```

```
export function setCurrency(currency) {
  return {
    type: WalletConstants.SET_CURRENCY,
    currency
  }
}

export function lockWallet() {
  return {
    type: WalletConstants.LOCK_WALLET
  }
}

export function unlockWallet() {
  return {
    type: WalletConstants.UNLOCK_WALLET
  }
}

export function unlockedWallet() {
  return {
    type: WalletConstants.UNLOCKED_WALLET
  }
}
```

13. App.js

```
import React, { Component } from 'react'
import { Switch, Route, Link } from 'react-router-dom'
import { connect } from 'react-redux'

import Console from './console'
import Identity from './identity'
import Versions from './_Versions'
import Init from './_Init'

import { init } from 'actions/Network'
import AccountChooser from 'components/AccountChooser'

import { selectAccount, setCurrency, loadWallet } from 'actions/Wallet'

class App extends Component {
  constructor(props) {
    super(props)
    this.state = {}
  }

  componentDidMount() {
    this.props.initNetwork()
  }
```

```
  componentWillUnmount() {
    clearTimeout(this.hideNotice)
  }

  componentDidUpdate(prevProps) {
    if (
      window.innerWidth <= 575 &&
      this.props.location !== prevProps.location
    ) {
      window.scrollTo(0, 0)
    }
  }

  componentWillReceiveProps(nextProps) {
    // If no accounts are present, pre-populate for an easier demo experience.
    if (
      !this.props.wallet.loaded &&
      nextProps.wallet.loaded &&
      !nextProps.wallet.activeAddress
    ) {
      window.sessionStorage.privateKeys = JSON.stringify([
        //   "0x24f3c3b01a0783948380fb683a9712f079e7d249c0461e1f40054b10b1bb0b23",  //  accounts[0]
ClaimSignerKey
        //TODO
        //  "0xd6079ba5123c57b9a8cb3e1fbde9f879c7a18eeca23fa2a965e8181d3ff59f0c",       //  accounts[1]
Identity
        // "0x20ea25d6c8d99bea5e81918d805b4268d950559b36c5e1cfcbb1cda0197faa08",     // accounts[2]
Certifier
        //  "0x25acb0da38f5364588f78b4e1f33c4a3981354c9b044d64bf201aad8f38f50ae",       //  accounts[3]
ClaimChecker
        '0x1aae4f8918c2c1fa3f911415491a49e541a528233da3a54df21e7eea5c675cd9',
        '0x7a8be97032a5c719d2cea4e4adaed0620e9fa9e49e2ccf689daf9180e3638f93',
        '0x85a676919234e90007b20bf3ae6b54b455b62b42bf298ac03669d164e4689c49'
      ])
      this.props.loadWallet()
      this.setState({ preloaded: true })
      this.hideNotice = setTimeout(
        () => this.setState({ preloaded: false }),
        3344
      )
    }
  }

  render() {
    return (
      <div>
        <Init onClose={() => this.props.history.push('/')} />
        <nav className="navbar navbar-expand-sm navbar-light">
          <div className="container">
```

```
    <Link
     to="/"
     className="navbar-brand mr-3"
     onClick={() => this.setState({ toggled: false })}
    >
     <a>Digital Identity ERC-725</a>
    </Link>

    <button
     className="navbar-toggler"
     type="button"
     onClick={() =>
      this.setState({
       toggled: this.state.toggled ? false : true
      })
     }
    >
     <span className="navbar-toggler-icon" />
    </button>
    <div
     className={`navbar-collapse collapse${
      this.state.toggled ? ' show' : ''
     }`}
    >
     <ul className="navbar-nav ml-auto text-right">
      {this.props.account &&
       this.props.wallet && (
        <li className="nav-item">
         <AccountChooser
          balance={this.props.balance}
          wallet={this.props.wallet}
          account={this.props.account}
          selectAccount={a => this.props.selectAccount(a)}
          setCurrency={c => this.props.setCurrency(c)}
         />
        </li>
       )}
     </ul>
    </div>
   </div>
  </nav>

  <div className="container">
   {!this.state.preloaded ? null : (
    <div className="alert alert-info mt-3">
     Logged in with demo account!
     <a
      className="close"
      href="#"
      onClick={e => {
```

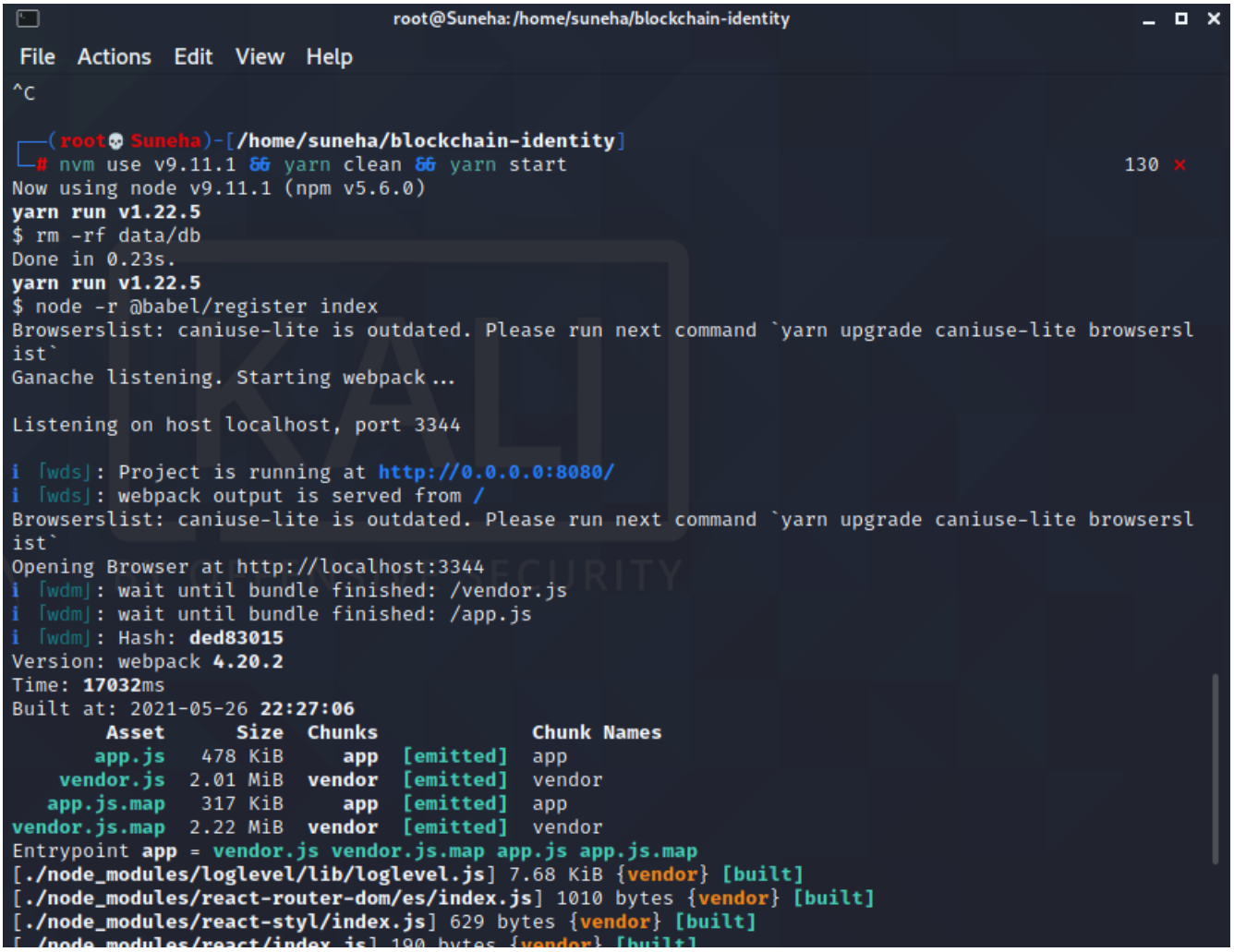
```
              e.preventDefault()
              this.setState({ preloaded: false })
            }}
          >
            &times;
          </a>
        </div>
      )}
      <Switch>
        <Route path="/console" component={Console} />
        <Route path="/identity/:address" component={Identity} />
        <Route path="/claim-checker/:address" component={Identity} />
        <Route component={Identity} />
      </Switch>
    </div>
  </div>
 )
 }
}

const mapStateToProps = state => ({
 account: state.wallet.activeAddress,
 balance: state.wallet.balances[state.wallet.activeAddress],
 wallet: state.wallet,
 nodeAccounts: state.network.accounts
})

const mapDispatchToProps = dispatch => ({
 initNetwork: () => {
  dispatch(init())
 },
 loadWallet: () => {
  dispatch(loadWallet())
 },
 selectAccount: hash => dispatch(selectAccount(hash)),
 setCurrency: currency => dispatch(setCurrency(currency))
})

export default connect(mapStateToProps, mapDispatchToProps)(App)

require('react-styl')(`
 table.table
  thead tr th
   border-top: 0
  .btn-sm
   padding: 0.125rem 0.375rem
 .navbar
  border-bottom: 1px solid #E5E9EF;
 .navbar-light .navbar-text .dropdown-item.active,
 .navbar-light .navbar-text .dropdown-item:active
```

```
    color: #fff;
  .pointer
    cursor: pointer
  .no-wrap
    white-space: nowrap
  .footer
    display: flex
    align-items: center;
    color: #999;
    margin: 1rem 0;
    padding-top: 1rem;
    border-top: 1px solid #eee;
    font-size: 14px;
    a
      color: #999;
    .middle
      flex: 1
      text-align: center
    .right
      flex: 1
      text-align: right
    .powered-by
      flex: 1
      font-size: 14px;
      letter-spacing: -0.01rem;
      img
        opacity: .4;
        height: 12px;
        margin-top: -2px
        margin-right: 0.25rem
`)
```

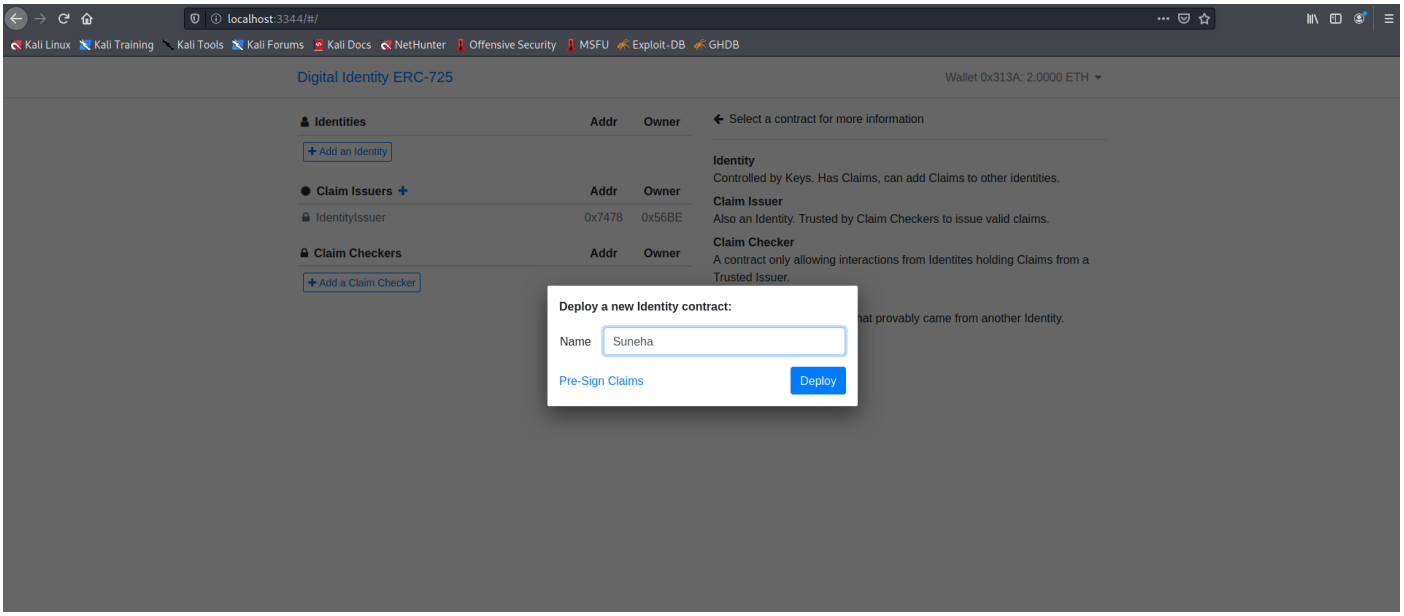
IMPLEMENTATION

1. We have implemented the complete project as root. For Starting the server we will first give the command

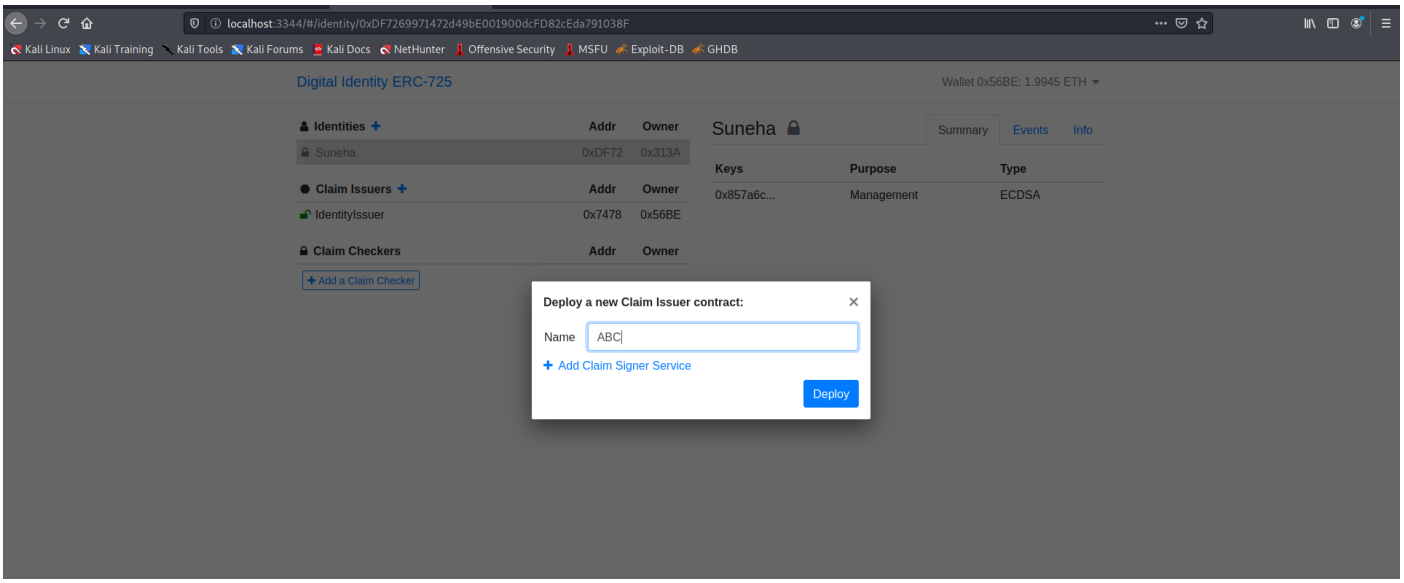
- nvm use 9.11.1 && yarn clean && yarn start



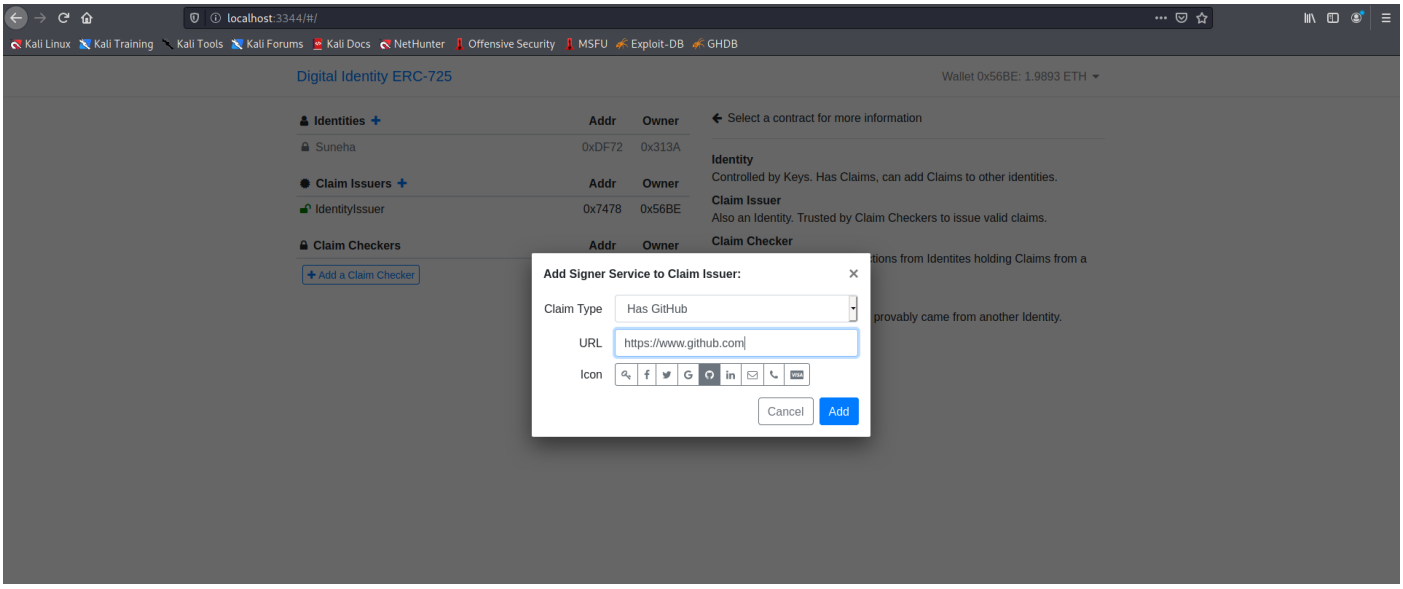
2. We can see the server host is http://localhost:3344
3. In the home page we can see Identities, Claim Issuers and Claim checkers.
4. We will first add a identity i.e. the person who wants to buy the property. After giving a name we will deploy it. Meanwhile the buyer has a wallet ID 0x313A



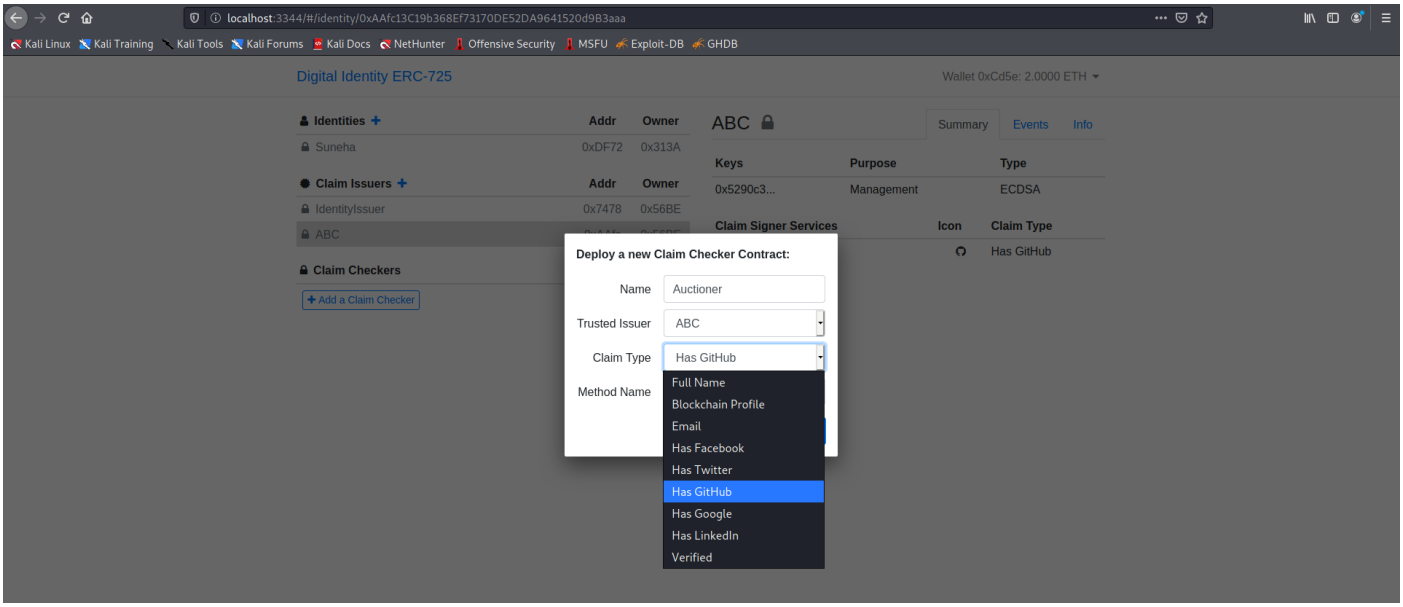
5. Then we will change the wallet to 0x56BE and add a claim issuer. The claim issuer who can verify the buyer's claim by issuing him claims.

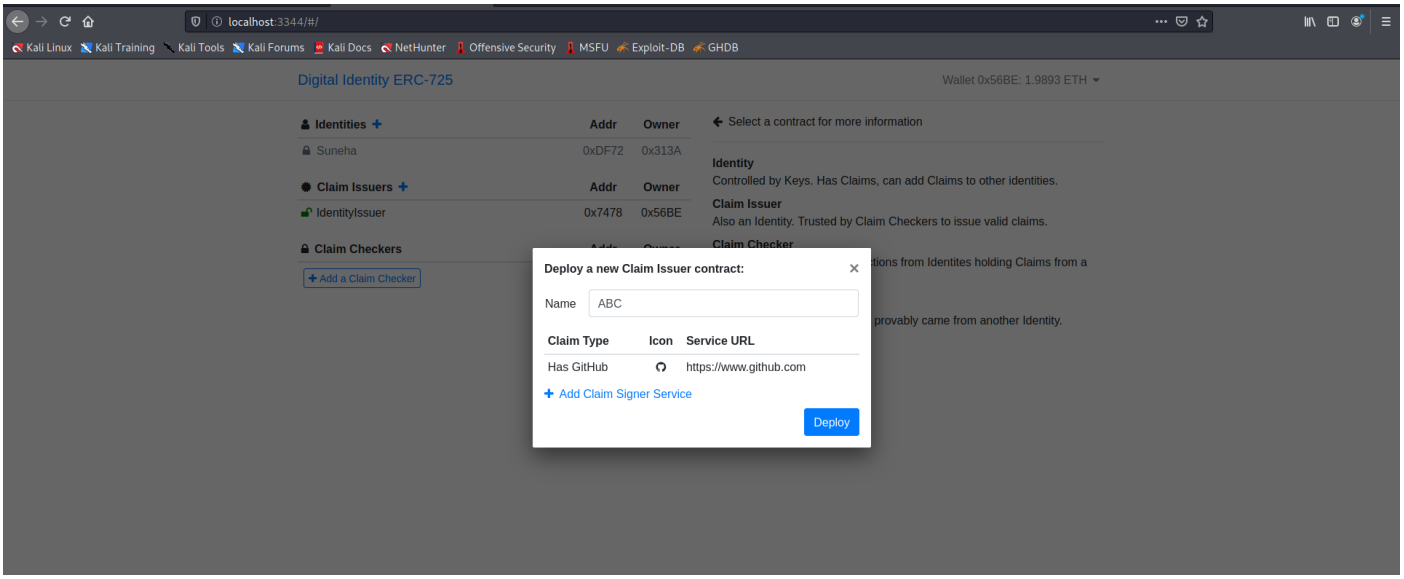


6. We then add claim signature to the issuer (here we are giving github as signature)

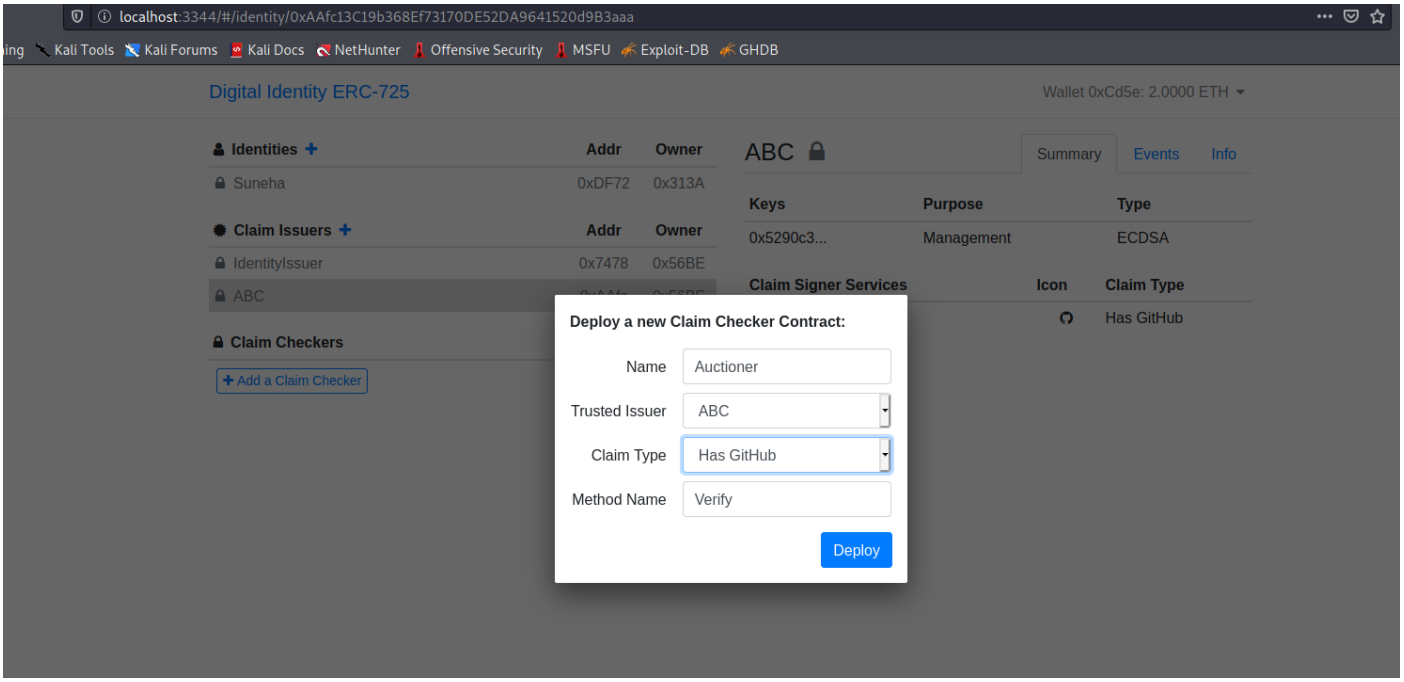


Here we have given ABC as the claim issuer

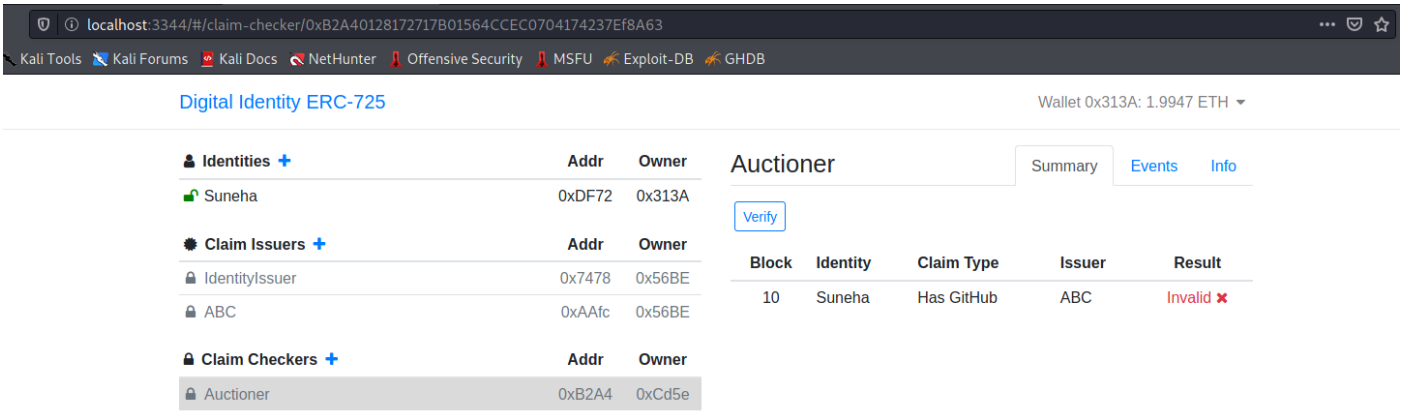




7. Again, change the Wallet to 0xCd5e and add a claim Checker i.e., the one who will verify the legitimacy of the claim made by the buyer. Here we give the checker name as auctioner and the trusted 3rd party issuer as ABC who can verify whether the buyer has a GitHub account or not.

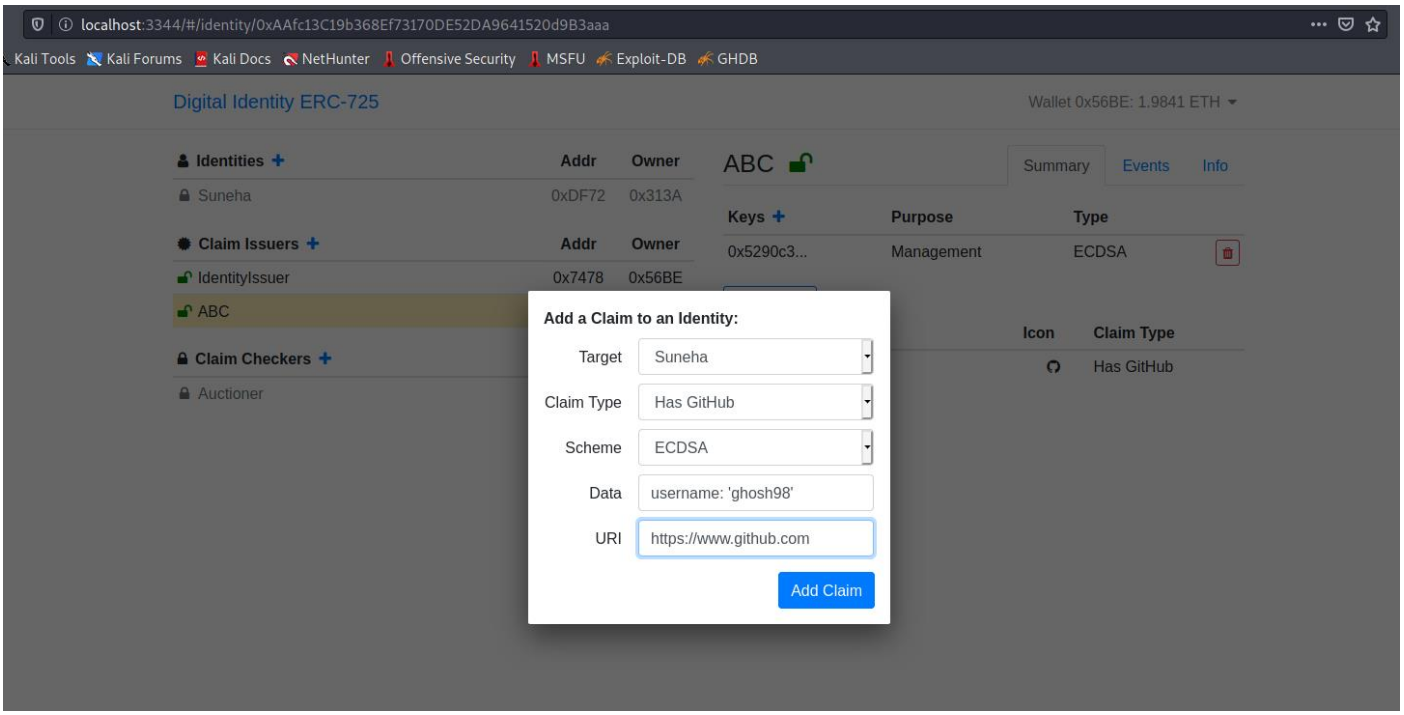


8. After this go to wallet id of the Identities and select Suneha
9. We have then added a self-claim of 'has GitHub' for the auctioner to see and verify.
10. Go to claim checker → Auctioner and click on verify. Due to the absence of the 3rd party claim issuer the self-claim of the buyer becomes invalid

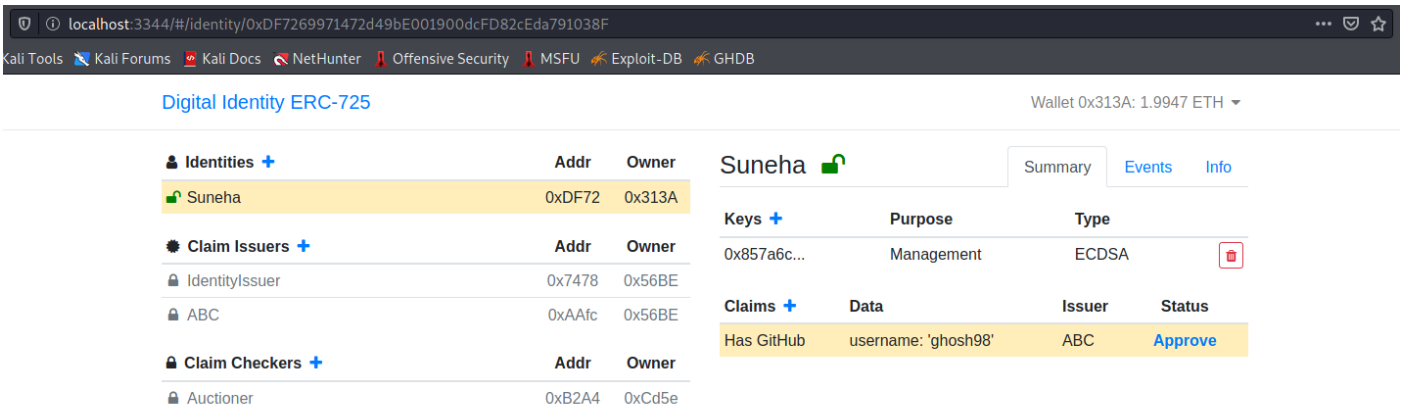


11. Then go to the 2nd wallet Id i.e. of the claim issuer and click on add a claim for an identity.

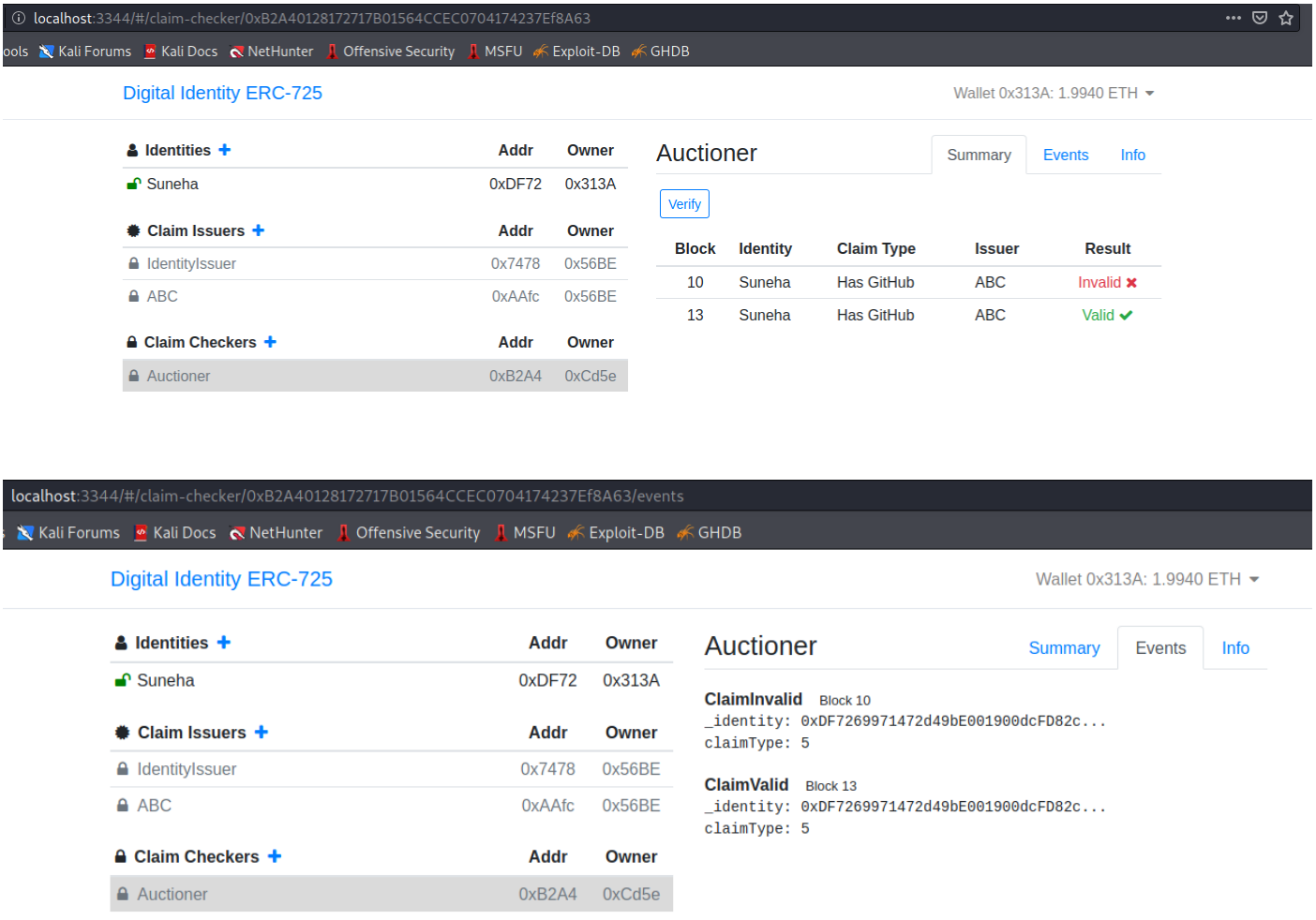
12. Select the identity (in this case suneha) and add a claim saying 'has GitHub'



13. Now go to the 1st wallet id i.e. of the identities and approve the claim.



14. Now go to the claim checker → Auctioner and click on verify. This time the claim will be shown as valid since the claim has been issued by a trusted 3rd party of the claim checker.



The claimType is 5 because in the code we have given it as type 5.

15. After the claim getting valid, the auctioner can sell the property to the buyer i.e. transaction can proceed without any doubt.

VIDEOLINK : Blockchain prerecorded video