BitTask

```
import { method, prop, SmartContract, hash256, assert, Sig,
PubKey, Utils, toByteString, hash160, FixedArray,
ContractTransaction, bsv } from 'scrypt-ts';
import { ZKProof } from 'scrypt-zk'; // Simulated zk for privacy
(sCrypt supports basic proofs)

export class TaskEscrow extends SmartContract {
    // ... existing props ...
    @prop(true)
    iotDevicePubKey: PubKey | null; // For IoT-assigned tasks

    // Constructor updated
    constructor(client: PubKey, freelancer: PubKey, arbitrator:
PubKey, taskHash: bigint, mneeAmount: bigint, iotDevicePubKey:
PubKey | null = null) {
        super(...arguments);
        // ... existing ...
        this.iotDevicePubKey = iotDevicePubKey;
    }

    @method()
    public complete(sig: Sig, deliverableHash: bigint, zkProof:
ZKProof) {
        assert(this.checkSig(sig, this.freelancer), 'Invalid signature');
        assert(zkProof.verify(deliverableHash, this.taskHash), 'Invalid
zk-proof'); // Privacy for deliverables
        this.isCompleted = true;

        // BRC-100 compute: IoT verification
```

```
        if (this.iotDevicePubKey) {
            assert(this.checkSig(sig, this.iotDevicePubKey), 'IoT device
must confirm');
            this.brc100State[3] = 1n; // IoT flag
        }

        const outputs =
Utils.buildPublicKeyHashOutput(hash160(this.freelancer),
this.mneeAmount); // $MNEE release
        assert(hash256(outputs) == this.ctx.hashOutputs, 'Output
mismatch');
    }

    // ... existing methods ...

    @method()
    public initiateDaoDispute(sig: Sig) {
        assert(this.isDisputed, 'No dispute');
        this.brc100State[4] = 1n; // Trigger BRC-100 DAO op
        this.buildStateOutput(this.ctx.utxo.value);
    }
}

import { method, prop, SmartContract, assert, Sig, PubKey,
hash256, Utils, Vote } from 'scrypt-ts';

export class ArbitrationDAO extends SmartContract {
    @prop(true)
    members: FixedArray<PubKey, 10>; // DAO members
    @prop(true)
    votes: FixedArray<Vote, 10>; // Votes for resolution
    @prop(true)
```

```
    threshold: bigint; // Majority threshold

    constructor(members: FixedArray<PubKey, 10>, threshold:
bigint) {
        super(...arguments);
        this.members = members;
        this.votes = Utils.fill({ voter: PubKey(''), decision: false }, 10);
        this.threshold = threshold;
    }

    @method()
    public vote(sig: Sig, index: bigint, decision: boolean, escrowTxId:
bigint) {
        const voter = this.members[Number(index)];
        assert(this.checkSig(sig, voter), 'Invalid voter');
        this.votes[Number(index)] = { voter, decision };

        // Tally votes via BRC-100 state compute
        let yesVotes = 0n;
        for (let i = 0n; i < 10n; i++) {
            if (this.votes[Number(i)].decision) yesVotes++;
        }

        if (yesVotes >= this.threshold) {
            // Resolve linked escrow
            const outputs =
Utils.buildOpReturnOutput(toByteString('resolve:' +
escrowTxId.toString() + ':' + decision)); // BRC-100 op
            assert(hash256(outputs) == this.ctx.hashOutputs, 'Output
mismatch');
        }
    }
```

```
}

import { method, prop, SmartContract, assert, Sig, PubKey,
hash256, Utils } from 'scrypt-ts';

export class ReputationNFT extends SmartContract {
    @prop(true)
    owner: PubKey;
    @prop(true)
    score: bigint;
    @prop(true)
    nftId: bigint; // BRC-100 mint ID

    constructor(owner: PubKey, nftId: bigint) {
        super(...arguments);
        this.owner = owner;
        this.score = 0n;
        this.nftId = nftId;
    }

    @method()
    public update(sig: Sig, delta: bigint) {
        assert(this.checkSig(sig, this.owner), 'Invalid signature');
        this.score += delta;
        this.buildStateOutput(this.ctx.utxo.value);
    }

    @method()
    public transfer(sig: Sig, newOwner: PubKey) {
        assert(this.checkSig(sig, this.owner), 'Invalid signature');
        this.owner = newOwner;
        this.buildStateOutput(this.ctx.utxo.value);
```

```
    }
}

// ... existing ...
const deployInscription = {
  p: 'BRC-100',
  op: 'deploy',
  tick: 'BitTask',
  max: '100000000',
  lim: '1000',
  ttp: '0.01',
  tr: 'your-wallet-address',
  ids: 'true', // DAO enabled
  iot: 'true' // New: IoT extension
};

// Add mint op for reputation NFTs
async function mintReputationNFT(privateKey, userPubKey, score)
{
  const inscription = {
    p: 'BRC-100',
    op: 'mint2',
    tick: 'BitTask',
    amt: '1',
    cop: `reputation:${userPubKey}:${score}` // BRC-100 compute
op
  };
  // Inscribe and broadcast...
}

// Similar for DAO vote op: op: 'compute', cop:
'dao_vote:escrowId:decision'
```

```
const express = require('express');
const { bsv, Provider } = require('@bsv/sdk');
const { Scrypt, deployContract, callContract } = require('scrypt-ts');
const cors = require('cors');
const WebSocket = require('ws');
const app = express();
app.use(cors());
app.use(express.json());

const provider = new Provider('teranode-mainnet'); // Upgraded to
Teranode
const wss = new WebSocket.Server({ port: 8080 }); // Real-time
updates

// Broadcast task updates
wss.on('connection', ws => {
  ws.on('message', message => {
    // Handle subscriptions...
  });
});

// Deploy escrow with IoT
app.post('/deploy-task', async (req, res) => {
  // ... existing ...
  const { iotDevicePubKey } = req.body;
  const instance = new TaskEscrow(/* args */,
bsv.PublicKey.fromString(iotDevicePubKey));
  // Deploy...
});

// New: Deploy DAO
```

```
app.post('/deploy-dao', async (req, res) => {
  const { members, threshold, privateKey } = req.body;
  const ArbitrationDAO = Scrypt.loadContract('./src/contracts/
ArbitrationDAO.ts');
  const instance = new
ArbitrationDAO(members.map(bsv.PublicKey.fromString),
BigInt(threshold));
  const deployTx = await deployContract(instance, 0n,
bsv.PrivateKey.fromWIF(privateKey), provider);
  res.json({ txId: deployTx.id });
});

// Vote in DAO
app.post('/dao-vote', async (req, res) => {
  const { daoTxId, index, decision, escrowTxId, privateKey } =
req.body;
  const instance = await Scrypt.loadArtifact(daoTxId, provider);
  const callTx = await callContract(instance, 'vote',
[bsv.Sig.fromPrivateKey(bsv.PrivateKey.fromWIF(privateKey)),
BigInt(index), decision, BigInt(escrowTxId)]);
  res.json({ txId: callTx.id });
});

// Enterprise API: Bulk tasks
app.post('/enterprise/bulk-tasks', async (req, res) => {
  const { tasks } = req.body; // Array of tasks
  const txIds = [];
  for (const task of tasks) {
    // Deploy each in parallel
    const deployTx = await deployContract(/* ... */);
    txIds.push(deployTx.id);
  }
```

```
  res.json({ txIds });
});

// IoT endpoint: Register device
app.post('/iot/register', async (req, res) => {
  const { devicePubKey } = req.body;
  // Mint BRC-100 IoT token...
  res.json({ success: true });
});

// Shard queries for scalability
// Use Redis or similar for sharding (pseudo)
app.get('/tasks', async (req, res) => {
  // Query sharded DB anchored on-chain...
});

app.listen(3001, () => console.log('Backend on port 3001'));

import React, { useState, useEffect } from 'react';
import axios from 'axios';
import io from 'socket.io-client'; // For real-time (alt to WS)

const socket = io('http://localhost:3001');

function App() {
  const [tasks, setTasks] = useState([]);
  const [taskHash, setTaskHash] = useState('');
  // ... existing states ...

  useEffect(() => {
    socket.on('task-update', (update) => {
      setTasks(prev => [...prev, update]);
```

```
    });
  }, []);

  const createTask = async () => {
    // ... existing ...
  };

  const registerIoT = async () => {
    const devicePubKey = 'iot-pubkey';
    await axios.post('http://localhost:3001/iot/register',
{ devicePubKey });
    alert('IoT device registered');
  };

  return (
    <div>
      <h1>BitTask Dashboard</h1>
      <input placeholder="Task Hash" onChange={e =>
setTaskHash(e.target.value)} />
      {/* ... */}
      <button onClick={createTask}>Create Task</button>
      <button onClick={registerIoT}>Register IoT Device</button>
      <ul>{tasks.map(task => <li key={task.id}>{task.description}</
li>)}</ul>
      {/* Add bidding, disputes, reputation mint */}
    </div>
  );
}

export default App;
```