|  |
| --- |
| const crypto = require('crypto'); |
|  | const EC = require('elliptic').ec; |
|  | const ec = new EC('secp256k1'); |
|  | const debug = require('debug')('savjeecoin:blockchain'); |
|  |  |
|  | class Transaction { |
|  | /\*\* |
|  | \* @param {string} fromAddress |
|  | \* @param {string} toAddress |
|  | \* @param {number} amount |
|  | \*/ |
|  | constructor(fromAddress, toAddress, amount) { |
|  | this.fromAddress = fromAddress; |
|  | this.toAddress = toAddress; |
|  | this.amount = amount; |
|  | this.timestamp = Date.now(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Creates a SHA256 hash of the transaction |
|  | \* |
|  | \* @returns {string} |
|  | \*/ |
|  | calculateHash() { |
|  | return crypto.createHash('sha256').update(this.fromAddress + this.toAddress + this.amount + this.timestamp).digest('hex'); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Signs a transaction with the given signingKey (which is an Elliptic keypair |
|  | \* object that contains a private key). The signature is then stored inside the |
|  | \* transaction object and later stored on the blockchain. |
|  | \* |
|  | \* @param {string} signingKey |
|  | \*/ |
|  | signTransaction(signingKey) { |
|  | // You can only send a transaction from the wallet that is linked to your |
|  | // key. So here we check if the fromAddress matches your publicKey |
|  | if (signingKey.getPublic('hex') !== this.fromAddress) { |
|  | throw new Error('You cannot sign transactions for other wallets!'); |
|  | } |
|  |  |
|  |  |
|  | // Calculate the hash of this transaction, sign it with the key |
|  | // and store it inside the transaction obect |
|  | const hashTx = this.calculateHash(); |
|  | const sig = signingKey.sign(hashTx, 'base64'); |
|  |  |
|  | this.signature = sig.toDER('hex'); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Checks if the signature is valid (transaction has not been tampered with). |
|  | \* It uses the fromAddress as the public key. |
|  | \* |
|  | \* @returns {boolean} |
|  | \*/ |
|  | isValid() { |
|  | // If the transaction doesn't have a from address we assume it's a |
|  | // mining reward and that it's valid. You could verify this in a |
|  | // different way (special field for instance) |
|  | if (this.fromAddress === null) return true; |
|  |  |
|  | if (!this.signature || this.signature.length === 0) { |
|  | throw new Error('No signature in this transaction'); |
|  | } |
|  |  |
|  | const publicKey = ec.keyFromPublic(this.fromAddress, 'hex'); |
|  | return publicKey.verify(this.calculateHash(), this.signature); |
|  | } |
|  | } |
|  |  |
|  | class Block { |
|  | /\*\* |
|  | \* @param {number} timestamp |
|  | \* @param {Transaction[]} transactions |
|  | \* @param {string} previousHash |
|  | \*/ |
|  | constructor(timestamp, transactions, previousHash = '') { |
|  | this.previousHash = previousHash; |
|  | this.timestamp = timestamp; |
|  | this.transactions = transactions; |
|  | this.nonce = 0; |
|  | this.hash = this.calculateHash(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Returns the SHA256 of this block (by processing all the data stored |
|  | \* inside this block) |
|  | \* |
|  | \* @returns {string} |
|  | \*/ |
|  | calculateHash() { |
|  | return crypto.createHash('sha256').update(this.previousHash + this.timestamp + JSON.stringify(this.transactions) + this.nonce).digest('hex'); |
|  | } |
|  |  |
|  |  |