"use strict";

// required npm install blind-signatures

const blindSignatures = require('blind-signatures');

const { Coin, COIN\_RIS\_LENGTH, IDENT\_STR, BANK\_STR } = require('./coin.js');

const utils = require('./utils.js');

// Details about the bank's key.

const BANK\_KEY = blindSignatures.keyGeneration({ b: 2048 });

const N = BANK\_KEY.keyPair.n.toString();

const E = BANK\_KEY.keyPair.e.toString();

/\*\*

\* Function signing the coin on behalf of the bank.

\*

\* @param blindedCoinHash - the blinded hash of the coin.

\*

\* @returns the signature of the bank for this coin.

\*/

function signCoin(blindedCoinHash) {

return blindSignatures.sign({

blinded: blindedCoinHash,

key: BANK\_KEY,

});

}

/\*\*

\* Parses a string representing a coin, and returns the left/right identity string hashes.

\*

\* @param {string} s - string representation of a coin.

\*

\* @returns {[[string]]} - two arrays of strings of hashes, commiting the owner's identity.

\*/

function parseCoin(s) {

let [cnst,amt,guid,leftHashes,rightHashes] = s.split('-');

if (cnst !== BANK\_STR) {

throw new Error(`Invalid identity string: ${cnst} received, but ${BANK\_STR} expected`);

}

let lh = leftHashes.split(',');

let rh = rightHashes.split(',');

return [lh,rh];

}

/\*\*

\* Procedure for a merchant accepting a token. The merchant randomly selects

\* the left or right halves of the identity string.

\*

\* @param {Coin} coin - the coin that a purchaser wants to use.

\*

\* @returns {[String]} - an array of strings, each holding half of the user's identity.

\*/

function acceptCoin(coin) {

// 1) Verify that the signature is valid.

const isValid = blindSignatures.verify({

unblinded: coin.signature,

N: coin.n,

E: coin.e,

message: coin.toString()

});

if (!isValid) {

throw new Error("Coin signature is invalid.");

}

// 2) Randomly select left or right RIS

const isLeft = utils.randInt(2) === 0;

const [leftHashes, rightHashes] = parseCoin(coin.toString());

const ris = [];

for (let i = 0; i < COIN\_RIS\_LENGTH; i++) {

const piece = coin.getRis(isLeft, i);

const hashCheck = utils.hash(piece);

const expectedHash = isLeft ? leftHashes[i] : rightHashes[i];

if (hashCheck !== expectedHash) {

throw new Error(`RIS piece at index ${i} does not match its hash.`);

}

ris.push(piece.toString('hex'));

}

return ris;

}

/\*\*

\* If a token has been double-spent, determine who is the cheater

\* and print the result to the screen.

\*

\* If the coin purchaser double-spent their coin, their anonymity

\* will be broken, and their idenityt will be revealed.

\*

\* @param guid - Globablly unique identifier for coin.

\* @param ris1 - Identity string reported by first merchant.

\* @param ris2 - Identity string reported by second merchant.

\*/

function determineCheater(guid, ris1, ris2) {

for (let i = 0; i < COIN\_RIS\_LENGTH; i++) {

const buf1 = Buffer.from(ris1[i], 'hex');

const buf2 = Buffer.from(ris2[i], 'hex');

const combined = Buffer.alloc(buf1.length);

for (let j = 0; j < buf1.length; j++) {

combined[j] = buf1[j] ^ buf2[j];

}

const combinedStr = combined.toString();

if (combinedStr.startsWith(IDENT\_STR)) {

console.log(`Double-spender detected! Coin ID: ${guid}, Purchaser identity: ${combinedStr.split(':')[1]}`);

return;

}

}

console.log(`Merchant cheated or reused RIS! Coin ID: ${guid}`);

}

// === Main Execution ===

let coin = new Coin('alice', 20, N, E);

coin.signature = signCoin(coin.blinded);

coin.unblind();

// Merchant 1 accepts the coin.

let ris1 = acceptCoin(coin);

// Merchant 2 accepts the same coin.

let ris2 = acceptCoin(coin);

// The bank realizes that there is an issue and

// identifies Alice as the cheater.

determineCheater(coin.guid, ris1, ris2);

console.log();

// On the other hand, if the RIS strings are the same,

// the merchant is marked as the cheater.

determineCheater(coin.guid, ris1, ris1);