Name	Kareem mohamed mohamed
Section	3
Code	220100399

Lab 5

```
"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 the signature.
  const valid = blindSignatures.verify({
    unblinded: coin.signature,
    N: coin.N,
    E: coin.E,
    message: coin.hashed
  });

  if (!valid) {
    throw new Error("Invalid coin signature.");
  }

  // 2) Randomly choose left or right half.
  const [leftHashes, rightHashes] = parseCoin(coin.toString());
  const ris = [];

  for (let i = 0; i < leftHashes.length; i++) {
    const useLeft = Math.random() < 0.5;
    const reveal = useLeft ? coin.identity[i][0] : coin.identity[i][1];
    const hash = useLeft ? leftHashes[i] : rightHashes[i];
    const computedHash = utils.hashString(reveal);

    if (computedHash !== hash) {
      throw new Error("Hash mismatch - coin may be tampered with.");
    }

    ris.push(reveal);
  }

  return ris;
}

/**
 * If a token has been double-spent, determine who is the cheater
 * and print the result to the screen.
 *
 * @param guid - Globally 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 < ris1.length; i++) {
    if (ris1[i] === ris2[i]) continue;

    let xorResult = '';
    for (let j = 0; j < ris1[i].length; j++) {
      xorResult += String.fromCharCode(ris1[i].charCodeAt(j) ^ ris2[i].charCodeAt(j));
    }

    if (xorResult.startsWith(IDENT_STR)) {
      const realId = xorResult.slice(IDENT_STR.length);
      console.log(Coin ${guid} was double-spent by user ${realId});
      return;
    } else {
      console.log(Coin ${guid} was reused fraudulently by a merchant.);
      return;
    }
  }

  console.log(Coin ${guid}: RIS strings are identical. Merchant is likely cheating.);
}

// ==========================
// Example 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 detects double spending and identifies Alice.
determineCheater(coin.guid, ris1, ris2);

console.log();

// If both RIS are the same, the merchant is the cheater.
determineCheater(coin.guid, ris1, ris1);
```