Summer University, Web Security Lab 3, July 2014

Name: Anushree, Denis, Israel, Raphael

Group: Calvin Hobbes

1. (10 points) This question deals with the RSA public key cryptosystem. Let p=3 and q=17, our "large" primes.

   (a) Calculate the modulus N with p and q. Use e=3. What is the public key?

   N = pq = 51    e = 3

   Public key: (N,e) = (51,3)

   (b) Find d such that

   $$ed \texttt{ mod } (p-1)(q-1) = 1$$

   This is your private key.

   (p-1)(q-1) = 32    e = 3

   ed mod 32 = 1

   (3)d mod 32 = 1

   Private key: d = 11

   (c) Encrypt the message $M = 26$ with Alice's public key, i.e. find $\{26\}_{Alice}$.

   M = 26
   C = M^e mod N = 26^3 mod 51 = 32

   (d) Decrypt the ciphertext $C = 18$ with Alice's private key. In other words, calculate $[18]_{Alice}$.

   M = C^d mod N = 18^11 mod 51 = 18

   (e) Assume that Trudy determines the value of p. What other details of the system can she now determine?

   She can determine the q and she can determine the private key.

2. (10 points) Install the keypair module for node:

```
npm install keypair
```

   (a) Download genKeys.js and testSig.js from the course website. Generate a public and private key. Update the `signMessage` and `verifySignature` methods from testSig.js. (You might find `http://nodejs.org/api/crypto.html` helpful). Turn in your modified testSig.js code. A sample run of this program's expected behavior is given below.

```
function signMessage(message, privKey, sigFile) {
  var signer = crypto.createSign('RSA-SHA256');
  signer.update(message)

  fs.writeFile(sigFile, signer.sign(privKey, "hex"), function(err) {
    if(err) { console.log(err); } else { console.log('Signature created'); }
  });
}
```

```
function verifySignature(message, pubKey, sig) {
  var verifier = crypto.createVerify('RSA-SHA256');
  verifier.update(message);
  return verifier.verify(pubKey, sig, "hex");
}
```

   (b) Download message.txt, sig.txt, alicePub.txt, bobPub.txt, and charliePub.txt.
   Who signed the message?

   Charlie was the original message signer.

3. (10 points) Create a first version of a cryptocurrency. Download cryptoCurrSimple.js, alicePriv.txt, alicePub.txt, bobPriv.txt, bobPub.txt. Update the `sendIOU` and `receiveIO` methods so that:

- The sender signs the message and includes the signature in the `transaction` object (in a field named `sig`).
- The receiver verifies the signature.

```
CoinClient.prototype.sendIOU = function(to, msg) {
  var signer = crypto.createSign('RSA-SHA256');
  signer.update(msg);
  var trans = {};
  trans.sig = signer.sign(this.privKey, "hex");
  trans.msg = msg;
  trans.pubKey = this.pubKey;
  trans.id = this.getID();
  var client = net.connect({port:to}, function() {
    client.write(JSON.stringify(trans));
  });
```

```
CoinClient.prototype.receiveIOU = function(client) {
  var verifier = crypto.createVerify('RSA-SHA256');

  client.on('data', function(data) {
    var trans = JSON.parse(data);
    console.log("Msg: " + trans.msg);
    verifier.update(trans.msg);
    if(verifier.verify(trans.pubKey, trans.sig, "hex"))
    { console.log("Signature Verified."); }
    if (rl) rl.prompt(); // Repeat prompt for UI
    }
  );
}
```

4. (10 points) Download cryptoCurrLedger.js. Update the `validateTransfer` method. Verify that the user has enough coins, and that the signatures are valid. Update the local copy of the ledger if everything seems valid.

```
CoinClient.prototype.validateTransfer =
function(trans) {
  var msg = JSON.stringify(trans.details);
  var coins = this.ledger[trans.id];

  //console.log(trans);
  var newLedger = {}; // Create new ledger
  newLedger.prev = this.ledger; // Retaining a copy of
the old ledger
  newLedger.next = this.ledger;

  //verify the signature
  var verifier = crypto.createVerify('RSA-SHA256');
  verifier.update(msg);

  if (verifier.verify(trans.pubKey, trans.sig, 'hex')) {
    var toUpdate = JSON.parse(msg);
    var checkSum = 0;
    for (var key in toUpdate) {
      if (toUpdate.hasOwnProperty(key)) {
          checkSum += toUpdate[key];
      }
    }
…
```

```
…
    if(checkSum <= coins) {
      newLedger.next[trans.id] = 0;
      for (var key in toUpdate) {
        if (toUpdate.hasOwnProperty(key)) {
          newLedger.next[key] += toUpdate[key];
        }
      }
    }

    this.ledger = newLedger.next;
    console.log("Verified Transaction: " + msg);
  } else  {
    this.ledger = newLedger.prev;
    console.log("Invalid Transaction: " + msg);
  }

  this.broadcast({type: 'accept'});
}
```

5. (10 points) Download proofOfWork.js from course website. Note that hash uses SHA256 to return the hash of a String as a binary String.

   (a) Implement the `findProof` function. This function should return a value proof such that

   `hash(value + proof)`

```
function findProof(s, numZeroes) {
  var i=0;
  var found = false;
  var hash_test;

  while(!found) {
    hash_test = hash(s + i);
    if(hash_test.indexOf("1", 0) >= numZeroes) {
      found = true;
      console.log(hash_test);
    } i++;
 } return i; }
```

   (b) How long does it take to find a proof for 2 leading zeroes? How about 10? 20?

   For finding 2 leading zeroes it takes less than a second to find the proof, as well as finding 10 leading zeroes. And for finding 20 zeroes it takes up to 4 seconds to find the proof.

   (c) Implement a `verifyProof` function that tests an alleged proof.

```
function verifyProof(s, numZeroes, proofVal) {
   var hash_test = hash(s + proofVal);
   if(hash_test.indexOf("1", 0) >= numZeroes)
{ return true; }
   return false;
}
```

6. (10 points) Combine the ledger-based cryptocurrency with your proof of work code.

   (a) Update the `validateTransfer` method to find a proof of work, where

   `hash(JSON.stringify(ledger+proof))`

   produces 10 leading zeroes. The ledger itself should include one additional "mined" cryptocoin for the miner's account (as the miner's reward for verifying the transaction). Broadcast your proof when found, and verify the proofs of others.

   (b) Add a `jsontransfer` command to the `readCommand` method. This method should work like `transfer`, except that it takes a JSON formatted string instead of prompting for users to specify coins. The object should specify how many coins each user will receive in the transaction. Validation should *not* be performed, since this will allow you to simulate an untrustworthy client.

   (c) (Bonus +5 points) Update your code so that it verifies the entire blockchain when a new proof is sent. If the new blockchain is shorter than the current one, ignore it. Likewise, if any block does not have a correct proof, ignore the new ledger.