SmileyCoin tutor-web assignments

Brynjar Ingimarsson

Task 3

In this example we show how we can prove our ownership of a base address. This is done by signing a message, resulting in a signature that others can verify.

Let's start by creating a new address to use in this example:

```
$ smileycoin-cli getnewaddress
BQozXBY6dd2zHJdKqvj13XoM7mHNS7b4zC
```

Now we sign a message using this address. This is done with the command *signmessage* with the following syntax

```
$ smileycoin-cli signmessage [address] [message]
```

We are going to sign the message "Hello World"

```
$ smileycoin-cli signmessage BQozXBY6dd2zHJdKqvj13XoM7mHNS7b4zC "Hello World"
H2ncRcee8kGRYMB0zrEwBabAskfvkp/Fxj4Yr1XV7PLq3H7b3tQe5abx/ne6W5RLVWHtVx6lOTW2
IUugSLWP1Hw=
```

Creating a signature requires having the private key that is behind our address, meaning that no one else could create this signature.

We could now tell people to verify that the message "Hello World" corresponds to our base address and this signature. This can be done with the command *verifymessage* in the wallet.

```
$ smileycoin-cli verifymessage [address] [signature] [message]
```

Let's verify the message we previously signed

```
$ verifymessage BQozXBY6dd2zHJdKqvj13XoM7mHNS7b4zC H2ncRcee8kGRYMB0zrEwBabAs
kfvkp/Fxj4Yr1XV7PLq3H7b3tQe5abx/ne6W5RLVWHtVx6lOTW2IUugSLWP1Hw= "Hello World"
true
```

The command returns true proving that this message came from us, thus proving our ownership of this address.

Q&A

Q: Is the following signature valid?

```
Message:   "Hello World"
Signature: IM+nbVpZUtghBACRy62TNa5+0I77+A+RriaN/9FOeSYdthT/cDJ6viH4NY9rYdJ9W+nZOG7HxRr34LiOeamT2wk=
Address:   BHdiMnruJtB9SifGefbEdq6yY473hsBoDh
```

A: Yes

This is a valid signature, we can see that by using the **verifymessage** command

```
$ smileycoin-cli verifymessage BHdiMnruJtB9SifGefbEdq6yY473hsBoDh IM+nbVpZ
UtghBACRy62TNa5+0I77+A+RriaN/9FOeSYdthT/cDJ6viH4NY9rYdJ9W+nZOG7HxRr34LiOeamT2wk= "Hello World"
true
```

Q: Is the following signature valid?

```
Message:   "Hello World"
Signature: H9rMpBOIoiaS+3e+eFxS8B45T3nEMvnkPnniuPGac9lG3MEVqmQDXyPi1LOl9FDVJ/6PJv6drQhSjV63MZNtgkY=
Address:   BHdiMnruJtB9SifGefbEdq6yY473hsBoDh
```

A: No

Task 4

In this task we show how to create a transaction that sends a part of an unspent transaction back to the same address, instead of creating a new one.

Here is an unspent transaction we will use for this example.

```
{
    "txid" : "eaa3da030a5e7b94f51aa319eca3875378ae1de975bf77bf947c838e032bf1
    48",
    "vout" : 1,
    "address" : "BCTw1xBdb6y1iqfVhWiPgPMnBJFqtR9Dt3",
    "account" : "",
    "scriptPubKey" : "76a91457ea3e6c75acf4203a3c84a92fa9230f0110391188ac",
    "amount" : 10000.00000000,
    "confirmations" : 1
}
```

You can see your unspent transactions by using

```
$ smileycoin-cli listunspent
```

I want to send 9900 SMLY to BEvkzqdQwhsPxDdsuFK8PVYtAxAbGxkreM. The easiest way would be to use

```
$ smileycoin-cli sendtoaddress [address] [amount]
```

But this would create a new unspent transaction with a new address. We want to keep the rest of our funds in the same address, so we will need to create a raw transaction.

```
./smileycoin-cli createrawtransaction
'[{
     "txid":"f419241c8aeae838390be2515a39c56d012744fc68287951266de36fa69deb1
f",
     "vout":1
}]'
'{
    "BCTw1xBdb6y1iqfVhWiPgPMnBJFqtR9Dt3":99,
    "BEvkzqdQwhsPxDdsuFK8PVYtAxAbGxkreM":9900
}'

010000000148f12b038e837c94bf77bf75e91dae785387a3ec19a31af5947b5e0a03daa3ea01
00000000ffffffff020003164e020000001976a91457ea3e6c75acf4203a3c84a92fa9230f011039
```

```
1188ac002c9980e60000001976a91472ed5cdf201eb86b466ed96e32c865dd25094aeb88ac000000
00
```

This returns the transaction in hex code. As always, we need to sign the transaction.

```
$ smileycoin-cli signrawtransaction 010000000148f12b038e837c94bf77bf75e91dae
785387a3ec19a31af5947b5e0a03daa3ea0100000000ffffffff020003164e020000001976a91457
ea3e6c75acf4203a3c84a92fa9230f0110391188ac002c9980e60000001976a91472ed5cdf201eb8
6b466ed96e32c865dd25094aeb88ac00000000
{
    "hex" : "010000000148f12b038e837c94bf77bf75e91dae785387a3ec19a31af5947b5
    e0a03daa3ea010000006a47304402203eb51038ddfef327fb7e0d420ef7396a990dfe702e874eae3
    0aac002b1d910f302207ea60529f2d9b5438f69b0fc5d5fcd1111c18750b3df4e2329755a8af17dc
    97b01210225da3a80b797627be0bfd94037b69d06b6eea6ce7059b2a269f2c8e688de8b01fffffff
    f020003164e020000001976a91457ea3e6c75acf4203a3c84a92fa9230f0110391188ac002c9980e
    60000001976a91472ed5cdf201eb86b466ed96e32c865dd25094aeb88ac00000000",
    "complete" : true
}
```

And then we send the transaction.

```
$ smileycoin-cli sendrawtransaction sendrawtransaction 010000000148f12b038e8
37c94bf77bf75e91dae785387a3ec19a31af5947b5e0a03daa3ea010000006a47304402203eb5103
8ddfef327fb7e0d420ef7396a990dfe702e874eae30aac002b1d910f302207ea60529f2d9b5438f6
9b0fc5d5fcd1111c18750b3df4e2329755a8af17dc97b01210225da3a80b797627be0bfd94037b69
d06b6eea6ce7059b2a269f2c8e688de8b01fffffff020003164e020000001976a91457ea3e6c75a
cf4203a3c84a92fa9230f0110391188ac002c9980e60000001976a91472ed5cdf201eb86b466ed96
e32c865dd25094aeb88ac00000000
8838d9bf1159ddf20ca46549eaab50c620194e069f6a1985810fc66618234b8e
```

We can look at the transaction in a blockchain explorer, where we see that the input address is also an output address.

https://chainz.cryptoid.info/smly/tx.dws?8838d9bf1159ddf20ca46549eaab50c620194e069f6a1985810fc66618234b8e.htm

Task 5

In this task we will try to find out how transaction fees work.

When we create a transaction using

```
$ smileycoin-cli sendtoaddress [address] [amount]
```

The wallet automatically adds 1 SMLY as transaction fee, this is the default fee as can be seen using **getinfo**

```
$ smileycoin-cli getinfo

...
    "paytxfee" : 1.00000000,
    "relayfee" : 0.00100000,
...
```

The transaction fee (paytxfee) is the minimum amount for miners to include your transaction in their blocks. If the fee is less than 1 SMLY your transaction will probably not make it into the blockchain.

The relay fee is the minimum transaction fee for other wallets to relay your transaction over the network.

Let's try to create a transaction with no transaction fee.

```
$ smileycoin-cli settxfee 0
$ smileycoin-cli sendtoaddress B7UmedvHCqRrjqqDjyYdefrwfKeaGc776y 10
```

We can see in a blockchain explorer that the transaction went through without a transaction fee. If the receiver types **listunspent** the amount can be seen

```
{
    "txid" : "1ca23532a98e4bcc801e6475ba1069b775cc7a7b13aacadb31436755c50aa3
    8a",
    "vout" : 1,
    "address" : "B7UmedvHCqRrjqqDjyYdefrwfKeaGc776y",
    "account" : "",
    "scriptPubKey" : "76a914213a361732bdbb6858b580fd38c0f10c01aeb12f88ac",
    "amount" : 10.00000000,
    "confirmations" : 2
},
```

I was not expecting this transaction to get picked up by any miner, but for some reason it did.

Task 6

In this task we will try to find out how the wallet adds data strings into transactions.

We can create a raw transaction with the following command

```
$ smly createrawtransaction '[{ "txid":"78ee210bc6a5d822dd2e5244e2f40d3df590
97afc539a1067cdb7a5ba46e733a", "vout":1 }]' '{ "B8Yna8SDPM8sx52uScCjWNGmDAxr4yp5
NV":53.5 }'
```

This creates the following hex string that describes the transaction

```
01000000013a736ea45b7adb7c06a139c5af9790f53d0df4e244522edd22d8a5c60b21ee7801
00000000ffffffff018085e23e010000001976a9142cf4c552cb6393a6c6db30356d39aeaf0999d5
eb88ac00000000
```

Now let's create the same transaction, but add some data

```
$ smly createrawtransaction '[{ "txid":"78ee210bc6a5d822dd2e5244e2f40d3df590
97afc539a1067cdb7a5ba46e733a", "vout":1 }]' '{ "B8Yna8SDPM8sx52uScCjWNGmDAxr4yp5
NV":53.5, "data":"4d" }'
```

Note that the data string must be in hex format. Here we added *4d*, which is one byte and corresponds to 77 in decimal system and the letter M in ASCII. The resulting transaction hex string is

```
01000000013a736ea45b7adb7c06a139c5af9790f53d0df4e244522edd22d8a5c60b21ee7801
00000000ffffffff028085e23e010000001976a9142cf4c552cb6393a6c6db30356d39aeaf0999d5
eb88ac000000000000000036a014d00000000
```

This string is identical to the hex string when there was no data, except for the following part at the end

```
00000000036a014d00000000
```

Also, in the middle of the string, a *1* became a *2* (presumably an output count field)

Now, if we now try adding *4d 4d 4d* the last part becomes

```
00000000056a034d4d4d00000000
```

We can see how *4d 4d 4d* is encoded in the string. The question is what the bytes in front of that mean. First it was *03 6a 01* and now it is *05 6a 03*. We can assume that this is some kind of a byte counter. When we changed the data from 1 to 3 bytes, the first part changed from *03* to *05* and the last part from *01* to *03*. It's also worth noting that *6a* is the hex code for the *OP_RETURN* script opcode.

We can read the Bitcoin documentation to see the actual data structure

```
Hex          Field           Data
-----------------------------------
00000000     SMLY amount     0
05           Script length   5
6a034d4d4d   Script          OP_RETURN 03 4d 4d 4d
00000000     Lockcode        0
```

Task 7

Creating and sending from multisig addresses.

In this example we show how a group of three people can create a shared address, such that the funds in it can only be spent if two out of three people approve. This is called a multisig address.

Breki's public key

```
028587e4ac345c8ce734d75c560ab26ab4c96150a6a9eefce76c3d25b9165a30be
```

Valdi's public key

```
02024265c781c484ad8ad1d5f70f353935e52fb1082fa02a4a1ecc59d7eafc264a
```

Brynjar's public key

```
02f3f77ec20ef25c85efcedca9c99ba905cc0c9465cb4dbf71e88cc2eae1368e67
```

Now we create a multisig address from the 3 public keys with the following command. We can specify how many signatures are needed to spend from the multisig, in this case 2. After that comes a JSON array with the public keys

```
  addmultisigaddress 2 '[
  "02024265c781c484ad8ad1d5f70f353935e52fb1082fa02a4a1ecc59d7eafc264a",
"028587e4ac345c8ce734d75c560ab26ab4c96150a6a9eefce76c3d25b9165a30be",
"02f3f77ec20ef25c85efcedca9c99ba905cc0c9465cb4dbf71e88cc2eae1368e67"]
```

The resulting multisig address is

```
3C3MZFrc6Hvp9N3LJaxogCQdthrZPJn2Rc
```

Each of us now transfers 100 SMLY to this multisig address.

Brynjar sent funds to the multisig with the sendtoaddress command, with the following transaction id

```
17184867fc42ad32441c452f1ca9e9a2ad08279cf00bf28b74a8942df4908e9e
```

Spending from the multisig address

Breki now wants to spend the money. He creates a raw transaction with Brynjar's transaction as input, sends 9 SMLY to himself and the rest back to the multisig address.

```
createrawtransaction '[ {
"txid":"17184867fc42ad32441c452f1ca9e9a2ad08279cf00bf28b74a8942df4908e9e",
"vout":0} ]' '{"3C3MZFrc6Hvp9N3LJaxogCQdthrZPJn2Rc":90,
"B7uV1XEpvcvCSEGU8u4Uwcg5oBTrNbTLj9":9}'
```

Breki uses signrawtransaction to sign the transaction hex

```
{ "hex" :
"01000000019e8e90f42d94a8748bf20bf09c2708ada2e9a91c2f451c4432ad42fc674818170
0000000b500483045022100d90d752b4c94930a47f17c73f760b7bc8133efe7cb2ee2120ef1ea0ef
ab6494b02201caf968e565aafd5eea10fe7c414209cd428124b829bd8edc33f05682b42e4d7014c69
522102024265c781c484ad8ad1d5f70f353935e52fb1082fa02a4a1ecc59d7eafc264a21028587e
4ac345c8ce734d75c560ab26ab4c96150a6a9eefce76c3d25b9165a30be2102f3f77ec20ef25c85e
fcedca9c99ba905cc0c9465cb4dbf71e88cc2eae1368e6753aeffffffff02001a71180200000017a
9147189f742ecbdadda408ffa38d2359345939ed0db8700e9a435000000001976a91425e6b85417a
23843a475567101a040e40a12a69a88ac00000000",
"complete" : false }
```

But he gets complete: false, because it also needs to be signed by Brynjar or Valdimar, to get the 2/3 required signatures.

He sends the hex output to Brynjar who signs it again and uses sendrawtransaction to send the transaction.

```
{ "hex" :
"01000000019e8e90f42d94a8748bf20bf09c2708ada2e9a91c2f451c4432ad42fc674818170
0000000fdfe0000483045022100d90d752b4c94930a47f17c73f760b7bc8133efe7cb2ee2120ef1e
a0efab6494b02201caf968e565aafd5eea10fe7c414209cd428124b829bd8edc33f05682b42e4d70
1483045022100869675d3b556957dac55253d9503d22662407d240b98bf2ae5f150ac979c2401022
00c6a275bfb5639ba704c0ae5678b7df36098a1a55f5e5f7ae4ffadf2341a028f014c69522102024
265c781c484ad8ad1d5f70f353935e52fb1082fa02a4a1ecc59d7eafc264a21028587e4ac345c8ce
734d75c560ab26ab4c96150a6a9eefce76c3d25b9165a30be2102f3f77ec20ef25c85efcedca9c99
ba905cc0c9465cb4dbf71e88cc2eae1368e6753aeffffffff02001a71180200000017a9147189f74
2ecbdadda408ffa38d2359345939ed0db8700e9a435000000001976a91425e6b85417a23843a4755
67101a040e40a12a69a88ac00000000",
"complete" : true }
```

Now we have complete: true so we know that we have enough signatures.:

```
sendrawtransaction
01000000019e8e90f42d94a8748bf20bf09c2708ada2e9a91c2f451c4432ad42fc6748181700
000000fdfe0000483045022100d90d752b4c94930a47f17c73f760b7bc8133efe7cb2ee2120ef1ea
0efab6494b02201caf968e565aafd5eea10fe7c414209cd428124b829bd8edc33f05682b42e4d701
483045022100869675d3b556957dac55253d9503d22662407d240b98bf2ae5f150ac979c24010220
0c6a275bfb5639ba704c0ae5678b7df36098a1a55f5e5f7ae4ffadf2341a028f014c695221020242
65c781c484ad8ad1d5f70f353935e52fb1082fa02a4a1ecc59d7eafc264a21028587e4ac345c8ce7
34d75c560ab26ab4c96150a6a9eefce76c3d25b9165a30be2102f3f77ec20ef25c85efcedca9c99b
a905cc0c9465cb4dbf71e88cc2eae1368e6753aeffffffff02001a71180200000017a9147189f742
```

```
ecbdadda408ffa38d2359345939ed0db8700e9a435000000001976a91425e6b85417a23843a47556
7101a040e40a12a69a88ac00000000
```

We can see all of this in a blockchain explorer:

https://chainz.cryptoid.info/smly/address.dws?3C3MZFrc6Hvp9N3LJaxogCQdthrZPJn2Rc.html

Task 9

Creating and sending from multisig addresses.

When we include data in a transaction, the wallet adds an output with zero as am ount and a script string that includes OP_RETURN and the data.

```
$ smly createrawtransaction '[{ "txid":"78ee210bc6a5d822dd2e5244e2f40d3df590
97afc539a1067cdb7a5ba46e733a", "vout":1 }]' '{ "B8Yna8SDPM8sx52uScCjWNGmDAxr4yp5
NV":50, "data":"4d6f6e65792066726f6d204272796e6a6172" }'
```

I encoded "Money from Brynjar" as ASCII, in hexadecimal the string becomes **4d 6f 6e 65 79 20 66 72 6f 6d 20 42 72 79 6e 6a 61 72**. The output of the command above is

```
01000000013a736ea45b7adb7c06a139c5af9790f53d0df4e244522edd22d8a5c60b21ee7801
00000000ffffffff0200f2052a010000001976a9142cf4c552cb6393a6c6db30356d39aeaf0999d5
eb88ac00000000000000000146a124d6f6e65792066726f6d204272796e6a617200000000
```

We can see the data in the last part of the transaction hex

```
00000000000000000146a124d6f6e65792066726f6d204272796e6a617200000000
```

The zeroes in the front are the amount in SMLY, in the case of data the amount i s zero. The next part is a script string, it starts with **14** which is the len gth of the script (20 in decimal). Next comes **6a** which is the opcode for OP_RETURN, then **12** which is the length of the data (18 bytes in decimal), and f inally the 18 byte long string. The zeros in the end are the locktime of the tra nsaction.

Without OP_RETURN

We could for example encode data in the amount that we send. Let's say that afte r the decimal we will put a sequence of numbers 01, 02, ... which correspond to the letters in the english alphabet

```
A = 01
B = 02
C = 03
...
Z = 26
```

The string "SMLY" would correspond to **19 13 12 25** in this system. Let's send this string encoded in the amount:

```
$ smly sendtoaddress 3C3MZFrc6Hvp9N3LJaxogCQdthrZPJn2Rc 0.19131225
```

We can see the transaction in the blockchain:

https://chainz.cryptoid.info/smly/address.dws?3C3MZFrc6Hvp9N3LJaxogCQdthrZPJn2Rc.htm

Task 10

Add a command to the smileycoin-cli command set; test the command and send a pull request to github (to https://github.com/tutor-web/smileyCoin)

Solution

https://github.com/tutor-web/smileyCoin/pull/14

Task 11

Implement a new ATM-style feature, e.g. based on the ATM toolkit: monitor transactions and perform some action(s)

Solution

https://github.com/Ingimarsson/smlymailer