



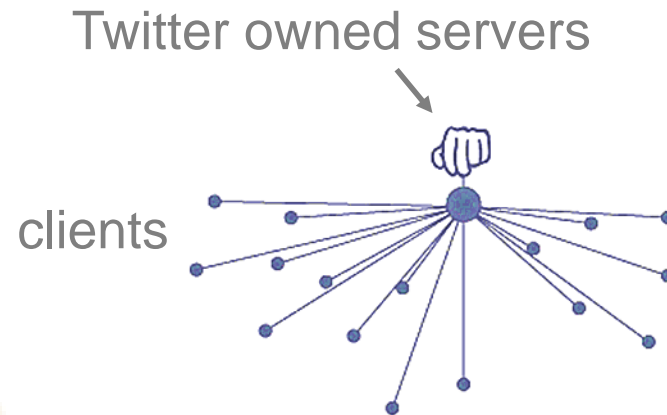
HOW TO WRITE **BITCOIN** **SMART CONTRACTS**

Feb 28, 2019
BitcoinPHL meetup

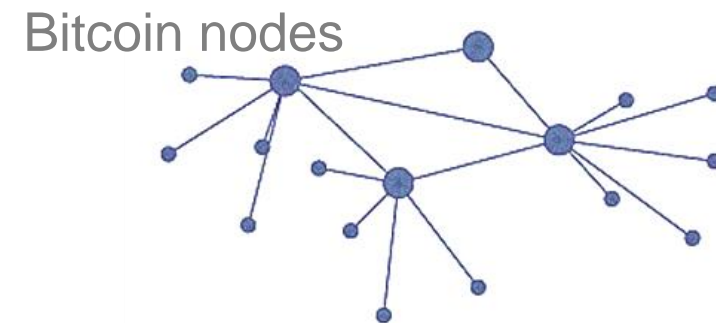
~15 minutes
ildarmgt@gmail.com

What is Bitcoin?

- Primary economic unit (**coins**) tracked on Bitcoin network
- Software protocols used to run the Bitcoin network
- Global public permissionless consensus network with **control** distributed to unlimited independent parties



Centralized
Twitter control

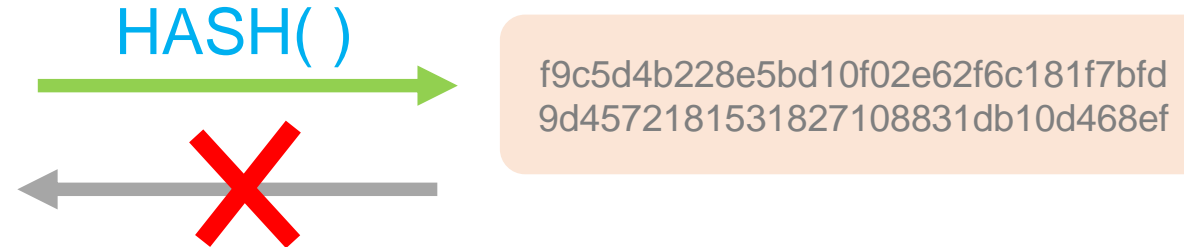


Decentralized
Bitcoin control

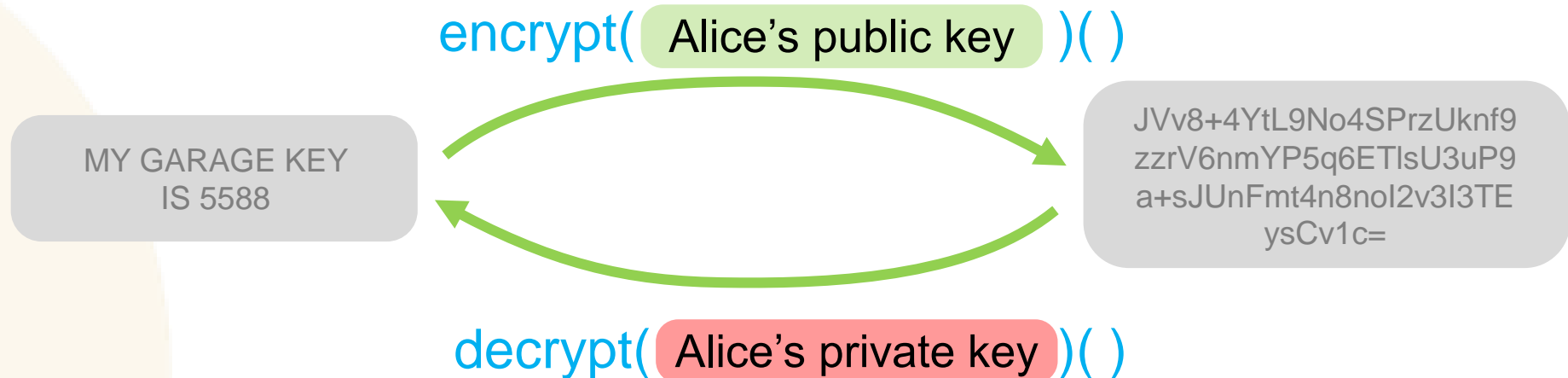
Bitcoin basics:

Hash function: irreversible conversion to unique fixed size data (often smaller)

Space: the final frontier. These are the voyages of the starship Enterprise. Its continuing mission: to explore strange new worlds. To seek out new life and new civilizations. To boldly go where no one has gone before!



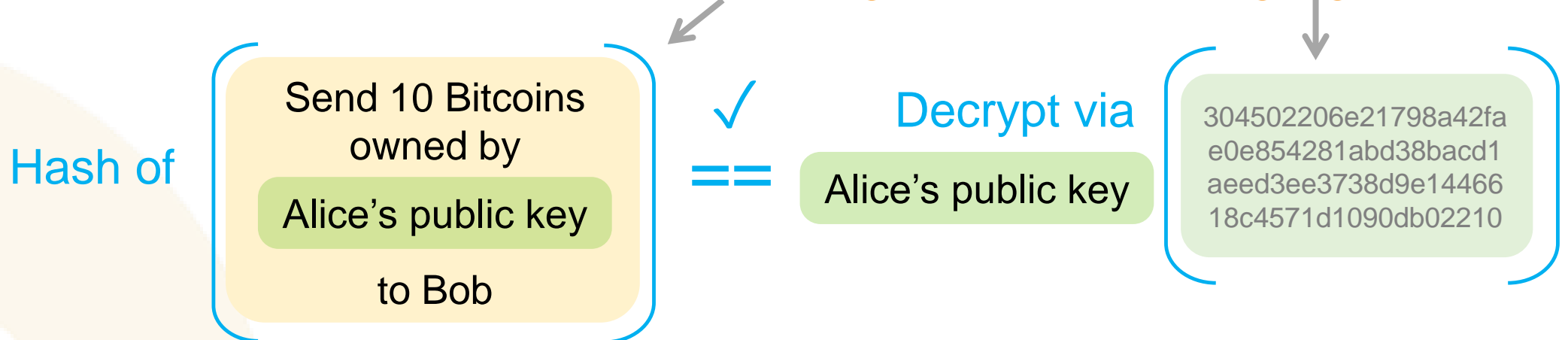
Encryption: reversible conversion. Uses a key or a key pair.



Signatures: prove a message was signed by a key pair owner



Alice's broadcast lets all see the message and its matching signature



Anyone can check message was signed by Alice's private key w/o seeing private key:

Classic Bitcoin protocol requires signature proof to allow spending

2.28.2019

BitcoinPHL

ildarmgt@gmail.com

What is a smart contract?

Computerized transaction **protocol** that executes the terms of a contract

- Nick Szabo (1994)

Observability: participants can **observe** any actions done to contract

Verifiability: protocol can **prove what happens** to participants

Privity: contract is **protected from unspecified third parties**

Szabo, N., First Monday, 1997, 2 (9)

Bitcoin is an example of
a popular smart contract!

Custom Smart Contracts on Bitcoin

Satoshi Nakamoto gave Bitcoin an interesting feature that wasn't described in the original whitepaper. Instead of requiring bitcoins be received to a public key and spent by a digital signature, Nakamoto gave users the ability to write programs (called scripts) that would act as dynamic public keys and signatures.



David A. Harding

Oct 12, 2017 · 11 min read



Sole participants specified via signature match

Conditions of contract

Bitcoin script example

```
OP_If  
  <Alice's pubkey> OP_CheckSig  
OP_Else  
  "3 months" OP_CSV OP_Drop  
  2 <Bob's pubkey> <Charlie's pubkey> 2 OP_CheckMultiSig  
OP_EndIf
```

2.28.2019

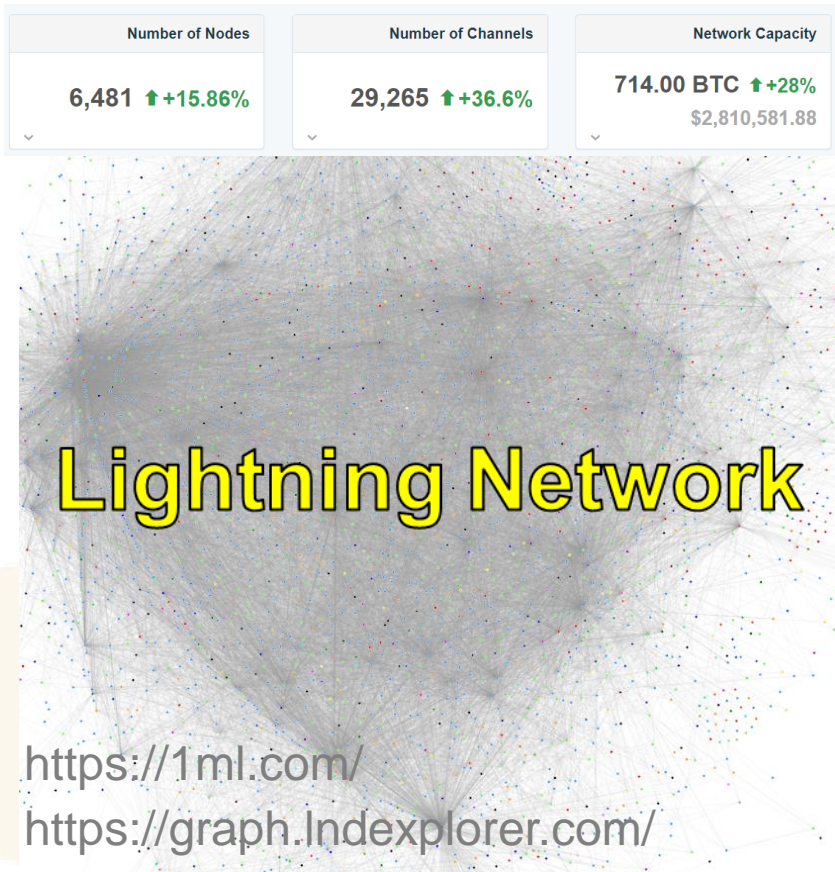
BitcoinPHL

ildarmgt@gmail.com

Opcodes descriptions e.g. <https://en.bitcoin.it/wiki/Script#Opcodes>

6 of 15

What has been done with Bitcoin scripts?



LN.PIZZA

2.28.2019
BitcoinPHL
ildarmgt@gmail.com

Example 1: Providing a deposit

Example 2: Escrow and dispute mediation

Example 3: Assurance contracts

Example 4: Using external state

6.1 Trust minimization: challenges

6.2 Trust minimization: multiple independent oracles

6.3 Trust minimization: trusted hardware

6.4 Trust minimization: Amazon AWS oracles

Example 5: Trading across chains (atomic swaps)

Example 6: Pay-for-proof contracts: buying a solution to any pure function

Example 7: Rapidly-adjusted (micro)payments to a pre-determined party

Example 8: Multi-party decentralised lotteries

Many other various examples online
(e.g. <https://en.bitcoin.it/wiki/Contract>)

Decide what you need the contract to do

“I want to create a contract for access to my Bitcoins with my key.
But, if I don’t move Bitcoins for 1 year,
my family inherits access with their own key”

Lets write it out in an easy to read pseudo-code

When claiming to be me:

Provide my signature or it fails

When claiming to be family:

Fail if 1 year has not passed.

Also provide family’s signature or it fails

Two parts to running Bitcoin script contracts

Unchangeable script

instructs execution of custom calculations on memory values

Examples:

if, add, check signature, verify if equal

Values in memory

come from the script or from user accessing the script

Examples:

5, “banana”, Alice’s signature

Values in memory are in a stack, similar to an array of items:

[5, “banana”, Alice’s signature, TRUE]

If script doesn’t fail early & last / top value is TRUE,
the script will allow spending

Let's walk through the script for me

MEMORY STACK

Me => mySignature, TRUE
mySignature, TRUE
mySignature

Bitcoin Contract

mySignature, <myPublicKey>
TRUE

At end of script, if top value is
TRUE, spending allowed

IF

<myPublicKey> CHECKSIG

ELSE

<1 year> CHECKSEQUENCEVERIFY

DROP

<famPublicKey> CHECKSIG

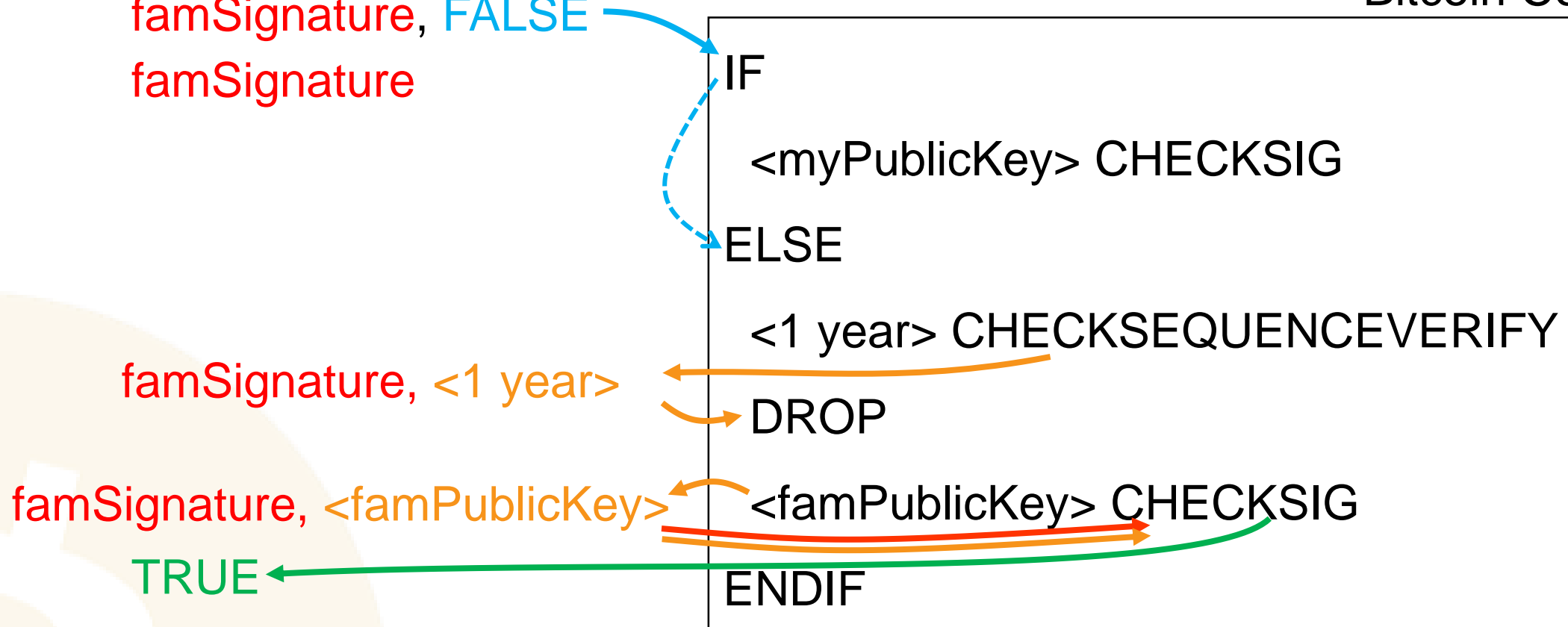
ENDIF

Let's walk through the script for my fam'

MEMORY STACK

family => famSignature, FALSE
famSignature, FALSE
famSignature

Bitcoin Contract



At end of script, top value is TRUE, so spending allowed

Easiest ways to start

Creating transactions from scratch is hard, so many rely on helpful libraries

Examples

Python : <https://github.com/petertodd/python-bitcoinlib>

C++ : <https://github.com/libbitcoin/libbitcoin-system>

Node.js : <https://github.com/bitcoinjs/bitcoinjs-lib>

JavaScript : <https://github.com/davidapple/bitcoinjs-lib-for-browsers>

There's always risk when using someone else's code or nodes.

Quality of randomness in implementations varies

For quick and dirty simple example demo
I'll use a JavaScript library and others' nodes

Intended for demo only!!!

Demo only using:

- Modern browser
- Text editor
- Internet

Calculate the contract address

(Code is on github)

1. Created `create.js`, `create.html` files and `libraries/` folder
2. Grabbed browser-friendly library files from github

```
└─ libraries
   ├── bitcoinjs-bip68.js
   ├── bitcoinjs-lib-4.0.2.min.js
   ├── create.html
   └── create.js
```

3. Filled out `create.html`

```
1 <html>
2   <head><meta charset="utf-8"></head>
3   <body>
4     <!-- ctrl-shift-j to see console -->
5     <!-- loading other people's libraries -->
6     <script src="libraries/bitcoinjs-lib-4.0.2.min.js"></script>
7     <script src="libraries/bitcoinjs-bip68.js"></script>
8     <!-- loading our script -->
9     <script src="create.js"></script>
10  </body>
11 </html>
```

Links `create.js` to `create.html`

4. Filled out `create.js`

```
1 // how long until inherit and keys to use (random keys here)
2 const YEARS_OF_DELAY = 1; // here must be (< 388 days)
3 const myKeyPair = bitcoin.ECPair.makeRandom();
4 const famKeyPair = bitcoin.ECPair.makeRandom();
5
6 // calculate relative lock time, has to be multiple of 512 seconds
7 const relativeLockTime = bitcoin.script.number.encode(bip68encode({
8   seconds: Math.floor(YEARS_OF_DELAY * 365.25 * 24 * 60 * 60 / 512) * 512
9 }));
10
11 // contract
12 const op = bitcoin.opcodes;
13 const redeemScript = bitcoin.script.compile([
14   op.OP_IF,
15   myKeyPair.publicKey, op.OP_CHECKSIG,
16   op.OP_ELSE,
17   relativeLockTime, op.OP_CHECKSEQUENCEVERIFY, op.OP_DROP,
18   famKeyPair.publicKey, op.OP_CHECKSIG,
19   op.OP_ENDIF
20 ]);
21
22 const p2sh = bitcoin.payments.p2sh({
23   redeem: { output: redeemScript, network: bitcoin.networks.bitcoin }
24 }).address;
25
26 console.log('BACK UP ALL THIS');
27 console.log('my private key:', myKeyPair.toWIF());
28 console.log('fam private key:', famKeyPair.toWIF());
29 console.log('script hex:', redeemScript.toString('hex'));
30 console.log('contract address:', p2sh);
```

Custom keys & delay

Bitcoin Script

Funding contract & Finding contract UTXO

- Opened [create.html](#) with Chrome & opened console (Ctrl-Shift-J)

```
BACK UP ALL THIS  
my private key: Kxy9aqev9aSuSRhiFp6KuMtYMW33z79ra8Es6  
fam private key: L2yqXBcwR2xULUJ4pXZbKvEDS8H6hhp7AaWk  
script hex:  
6321025aa85e44ef3e6b7c104eb9a378b5c3cbdbbe183f3f1d8ac  
contract address: 39JugGGhAEyUxbBGB4yWdgwm1S5c9B8ttX
```

- The new unspent output “UTXO” in the contract can be identified by this **txid** & **#0** vout

- Funded the **address** from random phone wallet

blockstream.info/address/39JugGGhAEyUxbBGB4yWdgwm1S5c9B8ttX

1 Transaction

Transaction id: **ce2acea70cd911c423d8c1007213761b1478d25052f30f71bdce56a76a6eebb5**

DETAILS +

#0	e6e63f1cf110721740ed44097397cfd709638dbf4e6ae	0.000392 BTC
#1	ea0450ec950dd6fc6b6e567cbb0255acdbf5af62923065a05946184a40fde8c1:1	0.00285329 BTC

vout

#0	39JugGGhAEyUxbBGB4yWdgwm1S5c9B8ttX	0.0005 BTC
#1	3Aq6WhGE1E7Ho1wMQuYKazGT4pTayWQDKW	0.00267105 BTC

Spending from P2SH output will need w/e script required like signing with private keys + output to spend (txid & vout) + entire original script itself

Code to spend the contract output (Code is on github)

Need new `spend.html` & `spend.js` files in same folder, linked like before

```
1 // SETTINGS: backups and spending choices
2 const MY_PRIVATE_KEY =
  'Kxy9aqev9aSuSRhiFp6KuMtYMW33z79ra8Es6iWcFmh83U2aNfAh';
3 const FAM_PRIVATE_KEY =
  'L2yqXBcwR2xULUJ4pXZbKvEDS8H6hhp7AaWK3mEH1djsFKq98CYi';
4 const SCRIPT_HEX =
  '6321025aa85e44ef3e6b7c104eb9a378b5c3cbdbbe183f3f1d8acb27bcf89ef0b265c1ac6703
  c3f040b2752102fe9bf1bf6afd6c2da3f0dd28c21eae01c5cf2ddba4202b37143ebd38a0abe5
  2ac68';
5 const SPEND_FROM_TXID =
  'ce2acea70cd911c423d8c1007213761b1478d25052f30f71bdce56a76a6eebb5';
6 const SPEND_FROM_VOUT = 0;
7 const UNSPENT_VALUE = 0.0005 * 1e8; // satoshi in output
8 const FEE_TO_PAY = 5000; // amount to spend on fee in satoshi
9 const WHERE_TO_SEND = '3JQtLGiuwPgYp5AW44b5so69gZR4oDVnSh'; // my wallet
```

```
28 // submit values and original script
29 const inputScriptMySpending = bitcoin.payments.p2sh({
30   redeem: {
31     input: bitcoin.script.compile([
32       mySignature,
33       bitcoin.opcodes.OP_TRUE
34     ]),
35     output: Buffer.from(SCRIPT_HEX, 'hex')
36   }
37 }).input;
38 tx.setInputScript(0, inputScriptMySpending);
39 console.log('Spend contract via this raw tx:', tx.toHex());
40 console.log('fee sat/byte:', (FEE_TO_PAY/tx.virtualSize()).toFixed(3));
```

Backed up data, from-to info, and amounts

Add values to submit & display raw tx

```
11 // build tx to sign for main net, 1 unspent input, 1 new output
12 const network = bitcoin.networks.bitcoin;
13 const buildTx = new bitcoin.TransactionBuilder(network);
14 buildTx.addInput(SPEND_FROM_TXID, SPEND_FROM_VOUT, 0xffffffff);
15 buildTx.addOutput(WHERE_TO_SEND, UNSPENT_VALUE - FEE_TO_PAY);
16 const tx = buildTx.buildIncomplete();
17
18 // calculate its hash and sign for w/e key
19 const hashType = bitcoin.Transaction.SIGHASH_ALL;
20 const hashForSignature = tx.hashForSignature(
21   0, Buffer.from(SCRIPT_HEX, 'hex'), hashType
22 );
23 const myKeyPair = bitcoin.ECPair.fromWIF(MY_PRIVATE_KEY, network);
24 const mySignature = bitcoin.script.signature.encode(
25   myKeyPair.sign(hashForSignature), hashType
26 );
```

Spend contract via this raw tx:
0200000001b5eb6e6aa756cebd710ff3
9022053b6dbb6a89925361e57331cf4a
b2752102fe9bf1bf6afd6c2da3f0dd28
fee sat/byte: 21.008

`spend.html` in Chrome

Broadcast raw transaction (hex)

0000000001b5eb6e6aa756cebd710ff35250d278141b76137200c1d823c411d90ca7
36d72900
01514c4f6321025aa85e44ef3e6b7c104eb9a378b5c3cbdbbe183f3f1d8acb27bcf89
0dd28c21eae01c5cf2ddba4202b37143ebd38a0abe52ac68feffff01c8af00000000
5b1e8700000000

Broadcast transaction

blockstream.info/tx/push

When ready, broadcast raw tx. Done.

Assembling & signing (standard steps)



- These slides
- All code used (demo use only & possible typos)

github.com/ildarmgt/client-side-bitcoin-contract-demo
take a picture if no QR code reader