lecture 15

discreet log contracts

2018-04-04

Tadge Dryja
today
discreet log contracts
conditional payments
oracles
anticipated signatures
building discreet log contracts
conditional payments

payment conditional on some external data

In this example, Alice and Bob bet on tomorrow's weather. If it rains, Alice gets 1 BTC. If it's sunny, Bob gets 1 BTC.

One problem: The bitcoin blockchain is not aware of the weather. (OP_WEATHER has not yet been soft-forked in)
"smart contracts" and oracles

LN is a simple script, enforcing the most recent tx

Made of smart contracts, but has no external state. Everything comes from Alice & Bob

If we want external state, need some way to get it, usually called an "oracle"

Simple oracle: 2 of 3 multisig
why oracles?

2 of 2 multisig means conflict freezes funds

Rich players at an advantage (lower time value of money)

Works great with friends, but bitcoin is the currency of enemies :)

A 3rd party can decide in case of conflict

2 of 3 multisig oracle
2 of 3 multisig oracle

3 keys: Alice, Bob, Olivia

If Alice and Bob are chill, they can both sign without contacting Olivia

If Alice and Bob fight or are unresponsive, one of them can ask Olivia to sign

Problem: It's sunny. Alice tells Olivia, "Hey, Alice. Say it's raining and I'll give you 0.8"
oracle interaction

2 of 3 multisig oracles are interactive

Not only do they see every contract, they decide the outcome of every contract, individually. (Can equivocate)

It'd be better if the oracle couldn't equivocate, and even better if they never saw the contracts. But how?
### revokable tx

<table>
<thead>
<tr>
<th>Commit Tx (held by Alice)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong></td>
</tr>
<tr>
<td>fund txid</td>
</tr>
<tr>
<td>Bob's signature</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### revokable tx

<table>
<thead>
<tr>
<th>Commit Tx (held by Bob)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong></td>
<td><strong>output</strong></td>
</tr>
<tr>
<td>fund txid</td>
<td>Alice address</td>
</tr>
<tr>
<td>Alice's signature</td>
<td>2 coins</td>
</tr>
<tr>
<td></td>
<td>Bob key &amp; 100 blocks</td>
</tr>
<tr>
<td></td>
<td>or Alice &amp; BobR key</td>
</tr>
<tr>
<td></td>
<td>8 coins</td>
</tr>
</tbody>
</table>
point and scalar operations
(Note also works on exponents mod n)
a, b lowercase = scalar
A, B uppercase = point
what operations can we do?
point and scalar operations

Scalars are regular unleaded numbers

\[ a+b \quad a-b \quad a*b \quad a/b \]

everything is OK! just numbers!
point and scalar operations

Points have addition defined... but not multiplication and division (group)

A+B  A-B  OK  A*B  A/B  NO

add & subtract OK, but can't multiply two points, or divide a point by a point. Not defined.
point and scalar operations

Mixed operations

A+b  A-b  NO  A*b  A/b  OK

adding points and scalars is undefined

point times scalar OK; repeat the
tangent doubling process. Division by
scalar also possible.
point and scalar operations
roster of ops: what can we do

\(a+b \ a-b \ a*b \ a/b \) (obvious)

\(A+B \ A-B \ A*b \ A/b\)
point and scalar operations

roster of ops: what can we do

a+b a-b a*b a/b (obvious)
A+B A-B A*b A/b

Pick some random point G
That's the generator point G
Everyone agrees on G
adding pubkeys

(aG) + (bG) = (a+b)G

sum of private keys gives sum of public keys! fun stuff ensues
adding pubkeys

\[ aG = A, \ bG = B \]

\[ A+B = C = (a+b)G \]

Alice knows a, Bob knows b. Neither can sign with C.

Bob can give b to Alice, then Alice can sign with C.
discreet log contracts
smart contracts in same channel
construction as lightning
lightning: most recent tx is valid
DLC: non-interactive oracle determines valid tx
schnorr signature

public key $A = aG$

$k \leftarrow \$; \ R = kG$ (nonce for signature)

to sign, compute $s = k - h(m, R)a$

signature is $(R, s)$

To verify $sG =? kG - h(m, R)aG =? R - h(m, R)A$
fixed-R signature

Pubkey: A signature: (R, s)

Pubkey: (A, R) signature: s

Same thing right? Just move the R. But can only sign once!
k-collision

Signature 1 \( s_1 = k - h(m_1, R)a \)

Signature 2 \( s_2 = k - h(m_2, R)a \)

\[ s_1 - s_2 = k - h(m_1, R)a - k + h(m_2, R)a \]

\[ = h(m_2, R)a - h(m_1, R)a \]

\[ = (h(m_2, R) - h(m_1, R))a \]

\[ a = \frac{s_1 - s_2}{h(m_2, R) - h(m_1, R)} \]

Fun fact: this is what brought down Playstation 3 code signing
anticipated signature

Given 'pubkey' (A, R) and a message m, you can't compute s.

(EC Discrete log problem)

but you CAN compute $sG = R - h(m, R)A$

$sG$ is computable for any message!
signatures as private keys
It's an unknown scalar, but you know what it is times the generator point. Hmm! Sounds like a keypair!
Use for a 3rd party oracle to sign messages, revealing a private key.
signatures as private keys

Olivia's $s$ as private key

$sG$ as public key

Mix with Alice and Bob's public keys

$$\text{pub}_{\text{alice}} + sG = \text{pub}_{\text{contract}}$$

$$\text{priv}_{\text{alice}} + s = \text{priv}_{\text{contract}}$$
In Lightning, states are added sequentially, and validity is enforced by revealing private keys to previous states.
signatures as private keys

In Lightning, states are added sequentially, and validity is enforced by revealing private keys to previous states.
signatures as private keys

In Lightning, states are added sequentially, and validity is enforced by revealing private keys to previous states.
In Lightning, states are added sequentially, and validity is enforced by revealing private keys to previous states.
In Lightning, states are added sequentially, and validity is enforced by revealing private keys to previous states.
signatures as private keys

In DLC all states are created at the start. Validity is determined by a non-interactive oracle signature.
signatures as private keys

In DLC all states are created at the start. Validity is determined by a non-interactive oracle signature.
Same script as LN

PubR OR (PubT AND time)

In lightning, The "correct" use is the timeout, op_csv

In cases of fraud, the revocable key can be used (half the key revealed)

In DLC, timeout is "incorrect", when someone publishes the wrong tx.
time and DLCs

In LN, you need to always watch for fraud, as old states could be broadcast. Gotta grab that output.

In DLC, you sweep the output as soon as you make it. Easier, and have the software broadcast both txs at the same time. No surprises.
DLCs within channels
Make a DLC output from an LN channel
If parties cooperate, 0 txs get broadcast to the blockchain
nested contracts

Fund txout A&B 30

DLC fund A&B 10

Sunny

Alice 5 BTC
Bob 15 BTC
Alice 1 BTC
Bob 9 BTC

Rainy

Alice 9 BTC
Bob 1 BTC

Oracle

Alice 5 BTC
Bob 15 BTC
Alice 1 BTC
Bob 9 BTC
Alice 9 BTC
Bob 1 BTC
nested contracts

Fund txout A&B 30

Alice 5 BTC
Bob 15 BTC

DLC fund A&B 10

State 1
Sunny

Alice 1 BTC
Bob 9 BTC

State 2
Rainy

Alice 9 BTC
Bob 1 BTC

Oracle
Sign(Rainy)
nested contracts

- Fund txout A&B 30
  - Alice 5 BTC
  - Bob 15 BTC
  - Oracle
    - Sign(Rainy)
  - DLC fund A&B 10
    - State 1
      - Sunny
        - Alice 1 BTC
        - Bob 9 BTC
    - State 2
      - Rainy
        - Alice 9 BTC
        - Bob 1 BTC
nested contracts

Fund txout A&B 30

DLC fund A&B 10

Alice 5 BTC

Bob 15 BTC

Oracle
Sign(Rainy)

State 2
Rainy

Alice 9 BTC

Bob 1 BTC
nested contracts

Fund txout A&B 30

Oracle
Sign(Rainy)

DLC fund A&B 10

Alice 5 BTC

Bob 15 BTC

Alice +9 BTC

Bob +1 BTC
nested contracts

Fund txout A&B 30

Alice 5 BTC
Bob 15 BTC
DLC fund A&B 10

Alice 14 BTC
Bob 16 BTC

Oracle
Sign(Rainy)
nested contracts

Fund txout A&B 30

Alice 5 BTC

Bob 15 BTC

DLC fund A&B 10

Alice 14 BTC

Bob 16 BTC

Oracle
Sign(Rainy)
DLC scalability

Can split the R value (and message) into a R-exponent and R-mantissa.

Helps cut down the off-chain transactions needed in ranges which don't lead to different allocations.
multi-oracle

Maybe Alice and Bob want to use 2 oracles. No problem.

\[ s_a G + s_b G = s_c G \]

Just add the \( sG \) points. \( n \) of \( n \), no size increase. (\( n \) of \( m \), size blowup)
DLC use cases
Currency futures? Stocks?
Commodities? Sports? Insurance?
Pretty general; conditional payments based on any number or element from predetermined set.