Atomic Swaps

coinfinity

Johannes Zweng

What is a “Swap”?

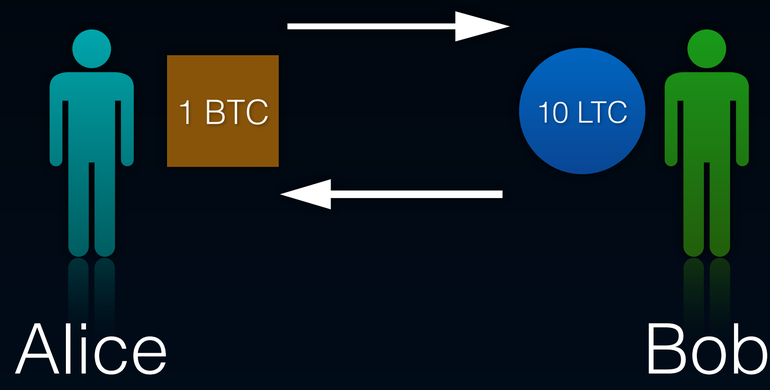
Other word for “Exchange”:

Example:

Alice has BTC
Bob has LTC

They negotiate with each other and both agree to exchange:

Alice wants to give Bob 1 BTC in return for 10 LTC

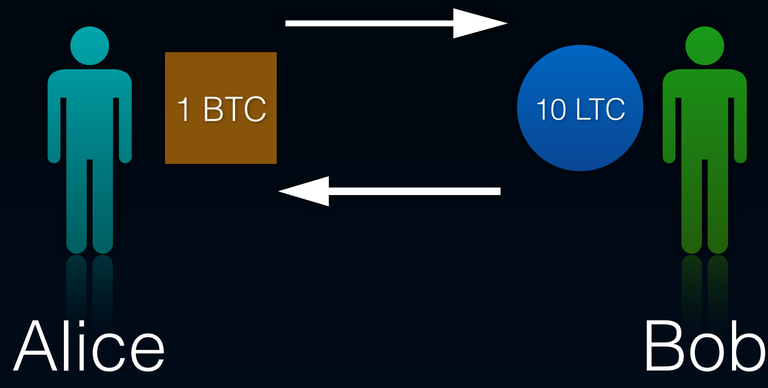


What is a “Swap”?

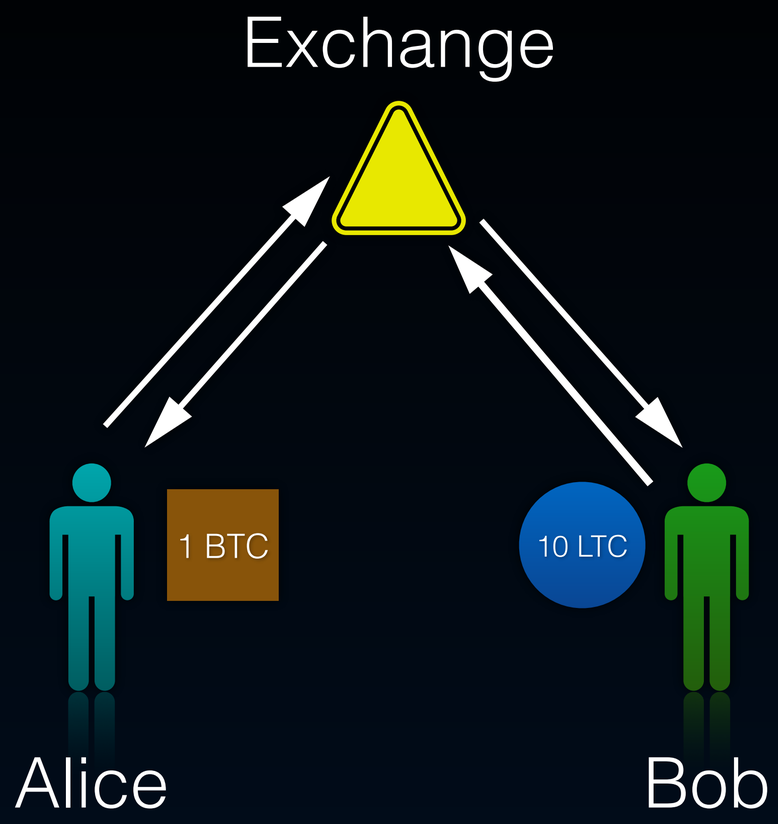
But how to make this secure?

If Alice sends her 1 BTC, how can she be sure, that she gets the 10 LTC from Bob?

Bob could just run away…



What is a “Swap”?



Current Situation:

- Need for trusted 3rd parties –> Centralized Exchanges

- What if…

  they somehow could exchange “*exactly at the same time*” ?

Atomic Swap

Atomicity:

> "*An **atomic transaction** is an **indivisible** […] **series** of […] **operations** such that **either all occur, or nothing** occurs.*"

Atomic Swap

- Basic Idea:
  Create transactions, on both chains
- Add a spending condition, which only can get true on both chains simultaneously (even if chains are totally unrelated)

Atomic Swap

- Already in 2011 Mike Hearn proposed an idea to achieve this (see Bitcoin Wiki, Article “Contract”, Example 5): *“Trading across chains”*

Atomic Swap

The basic idea is the following:

- Initiator (i.e. Alice) thinks of a random secret (“**S**”) example:

  “***correct horse battery staple***”

- She calculates the hash (“**H**”) of the secret “S”:

  “***2259cd5b42ae4d70deaa3d8d2ead2bb32ed3677b***”

Atomic Swap

- Then Alice sends her funds (1 BTC) into a “Contract TX” (or “Funding” TX) on the BTC chain, locking the output like this:

Alice's **BTC** Funding TX

1 BTC

"Funding" TX

signature Bob

+

secret "**S**" ("**correct horse bat…**")

(possible immediately)

This is 1 output, with 2 alternative spending conditions:

signature Alice

(possible after some time in future)

failsafe "refund" for Alice

OR

Going to Bob if he knows "S"

Going back to Alice

(after some time)

Alice's **BTC** Funding TX

1 Output (*ie. 1 BTC*)
Can be spent **EITHER**:

- script:

```
IF
   <Key_Bob> CHECKSIGVERIFY
   HASH160 <H> EQUALVERIFY
ELSE
   <Key_Alice> CHECKSIGVERIFY
   <Alice_Timeout> CHECKLOCKTIMEVERIFY
ENDIF
```

by Bob (*Key_Bob*) if he knows the secret "**S**" which will hash to the value "**H**"

OR:

by Alice (*Key_Alice*) at some time in future (failsafe refund)

Alice's **BTC** Funding TX

*This type of Tx are called "**HTLC**":*
*"**Hash-time-locked contract**"*

1 Output (*ie. 1 BTC*)
Can be spent **EITHER**:

- script:

```
IF
   <Key_Bob> CHECKSIGVERIFY
   HASH160 <H> EQUALVERIFY
ELSE
   <Key_Alice> CHECKSIGVERIFY
   <Alice_Timeout> CHECKLOCKTIMEVERIFY
ENDIF
```

by Bob (*Key_Bob*) if he knows the secret "**S**" which will hash to the value "**H**"

OR:

by Alice (*Key_Alice*) at some time in future (failsafe refund)

- spending input data:

```
1 <sig Bob> <secret S>
```

OR:

```
0 <sig Alice> (after some time)
```

Bob's **LTC** Funding TX

1 single Output (*ie. 10 LTC*)
Can be spent **EITHER**:

- script:

```
IF
   <Key_Alice> CHECKSIGVERIFY
   HASH160 <H> EQUALVERIFY
ELSE
   <Key_Bob> CHECKSIGVERIFY
   <Bob_Timeout> CHECKLOCKTIMEVERIFY
ENDIF
```

by Alice (*Key_Alice*) if she knows (and provides) the secret "**S**" which will hash to the value "**H**"

OR:

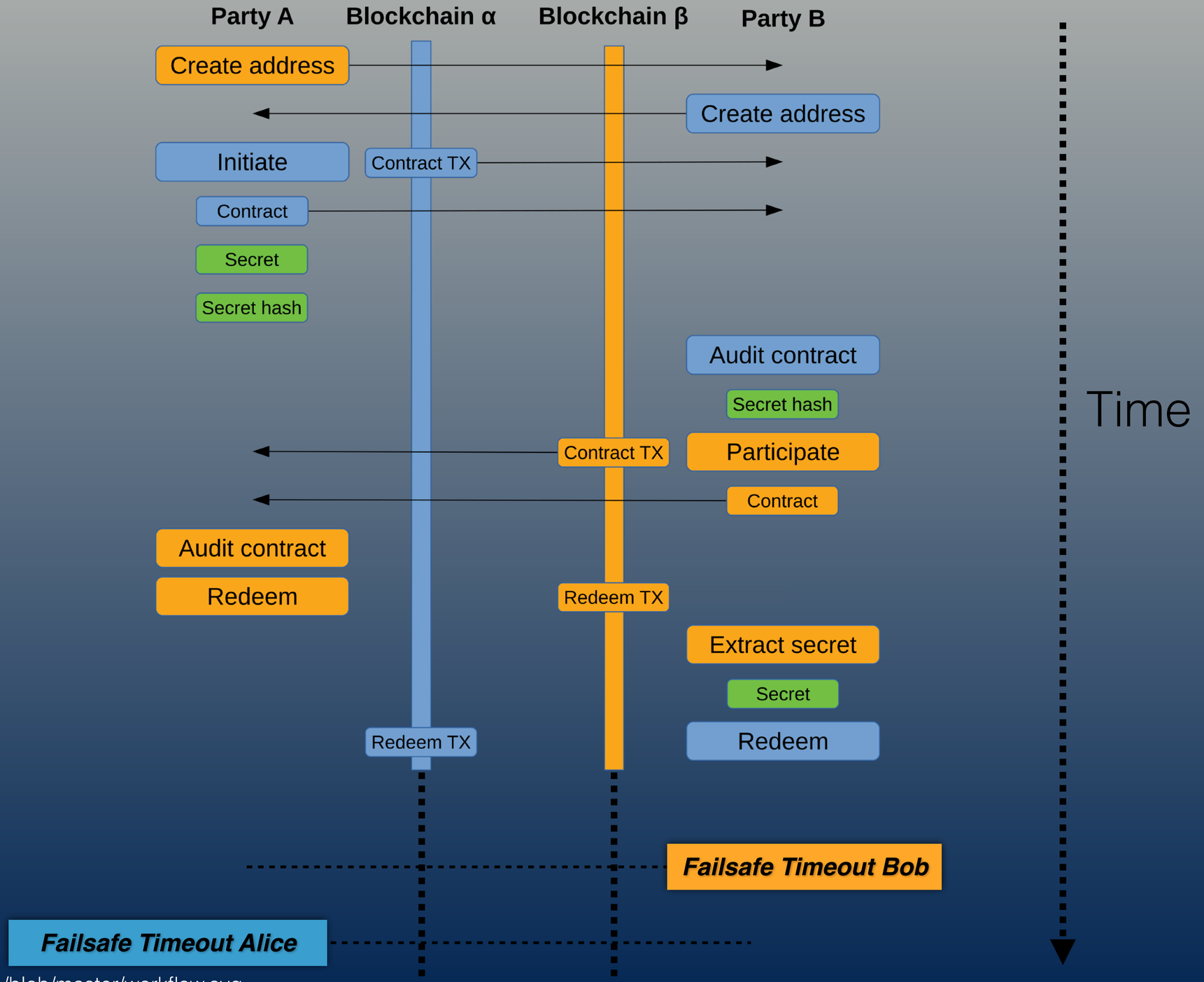
by Bob (*Key_Bob*) at some time in future (failsafe refund)

- spending input data:

```
1 <sig Alice> <secret S>
```

OR:

```
0 <sig Bob> (after some time)
```



Source:
https://github.com/decred/atomicswap/blob/master/workflow.svg

What do the 2 chains need?

- possibility to somehow time-lock funds (CLTV or CSV on Bitcoin-like chains)
- support the same hashing algorithm in the evaluating script
- branching support in scripts (if / else) to realize failsafe path
- ability to check hashes and signatures in evaluating script

What do the 2 chains need?

- this is true for most Bitcoin-like chains
- also for smart contract chains (which allow to program conditions totally free), i.e. Ethereum..
- not possible on chains which don’t allow to express spending conditions based on hash pre-image

Secret size attack

Remember, our secret:

- “***correct horse battery staple***”
  which hashes to:
  “***2259cd5b42ae4d70deaa3d8d2ead2bb32ed3677b***”

- Is there a limit for the maximum possible length of a secret?

- For Bitcoin: Yes! 520 Bytes (source)

```
20  // Maximum number of bytes pushable to the stack
21  static const unsigned int MAX_SCRIPT_ELEMENT_SIZE = 520;
```

- What if this limit is different between two chains?

source: https://gist.github.com/markblundeberg/7a932c98179de2190049f5823907c016, Dr. Mark B. Lundeberg, Feb 15, 2018

Secret size attack

Example:

- Imagine evil attacker “**Eve**” 😈 owns **FantasyCoin “FC”** which allows max. 300 bytes-sized script elements
- Eve and Alice agree to trade **10000 FC against 10 BTC**
- Eve creates a **secret** which is **>300 bytes but <520 bytes long** and hashes it into 160 bytes
- Eve proceeds as discussed before (locks her FC into the Funding TX, informs Alice)
- As soon as Alice has locked her 10 Bitcoin in her Funding TX, Eve can claim them (as planned, because she as initiator knows the secret)
- But when Alice now wants to claim her 10000 FC in return, she cannot: although she now knows the secret, she cannot use it, as it’s too large to be used in a FC coin script.

Secret size attack

Fortunately, there is an easy solution:

- Add conditon into the script which commits to the length of the secret
- For Bitcoin Script: add “**OP_SIZE xx OP_EQUALVERIFY**” commands. This way the size will be public beforehand:

```
IF
   <Key_Eve> CHECKSIGVERIFY
   SIZE 28 EQUALVERIFY
   HASH160 <H> EQUALVERIFY
ELSE
   <Key_Alice> CHECKSIGVERIFY
   <Bob_Timeout> CHECKLOCKTIMEVERIFY
ENDIF
```

Tools for on-chain swaps

- **“decred”**: https://github.com/decred/atomicswap/

- Commanline tools (for each supported chain) to create onchain atomic swap transactions and perform all steps in the protocol:

```
Commands:
  initiate <participant address> <amount>
  participate <initiator address> <amount> <secret hash>
  redeem <contract> <contract transaction> <secret>
  refund <contract> <contract transaction>
  extractsecret <redemption transaction> <secret hash>
  auditcontract <contract> <contract transaction>
```

```
$ btcatomicswap --testnet --rpcuser=user --rpcpass=pass initiate n31og5QGuS28dmHpDH6PQD5wmVQ2K2spAG 1.0
Secret:      3e0b064c97247732a3b345ce7b2a835d928623cb2871c26db4c2539a38e61a16
Secret hash: 29c36b8dd380e0426bdc1d834e74a630bfd5d111
```

- support several chains: **BTC, BCH, LTC, Monacoin, Particl, Polis, Vertcoin, Viacoin, Zcoin** (Ethereum contract currently WIP*: AtomicSwap.sol)*

Alternative protocol

- BarterDEX by jl777 (used in the Komodo platform): https://komodoplatform.com/decentralized-exchange/
- Realizes atomic swaps with key pairs chosen using the “Cut and choose” principle, as proposed by TierNolan in bitcointalk forum in 2016: https://bitcointalk.org/index.php?topic=1340621.msg13828271#msg13828271 and https://bitcointalk.org/index.php?topic=1364951
- Whitepaper: https://github.com/KomodoPlatform/KomodoPlatform/wiki/barterDEX-Whitepaper-v2
- More complex, takes 7 steps, needs deposits as incentive, needs percental fees to mitigate incentive to exploit the cut-and-choose process, …). But works also if one of the 2 chains has no support for CHECKLOCKTIMEVERIFY
- https://dexstats.info/

Same-chain token swaps

For the sake of completeness:
Not cross-chain, but these are also examples of atomic swaps:

- many ICO contracts on the Ethereum blockchain (atomically exchange ETH against tokens)
- Onchain ERC-20 Exchanges (like EtherDelta, IDEX, 0x, etc..) atomically exchange ERC-20 Tokens

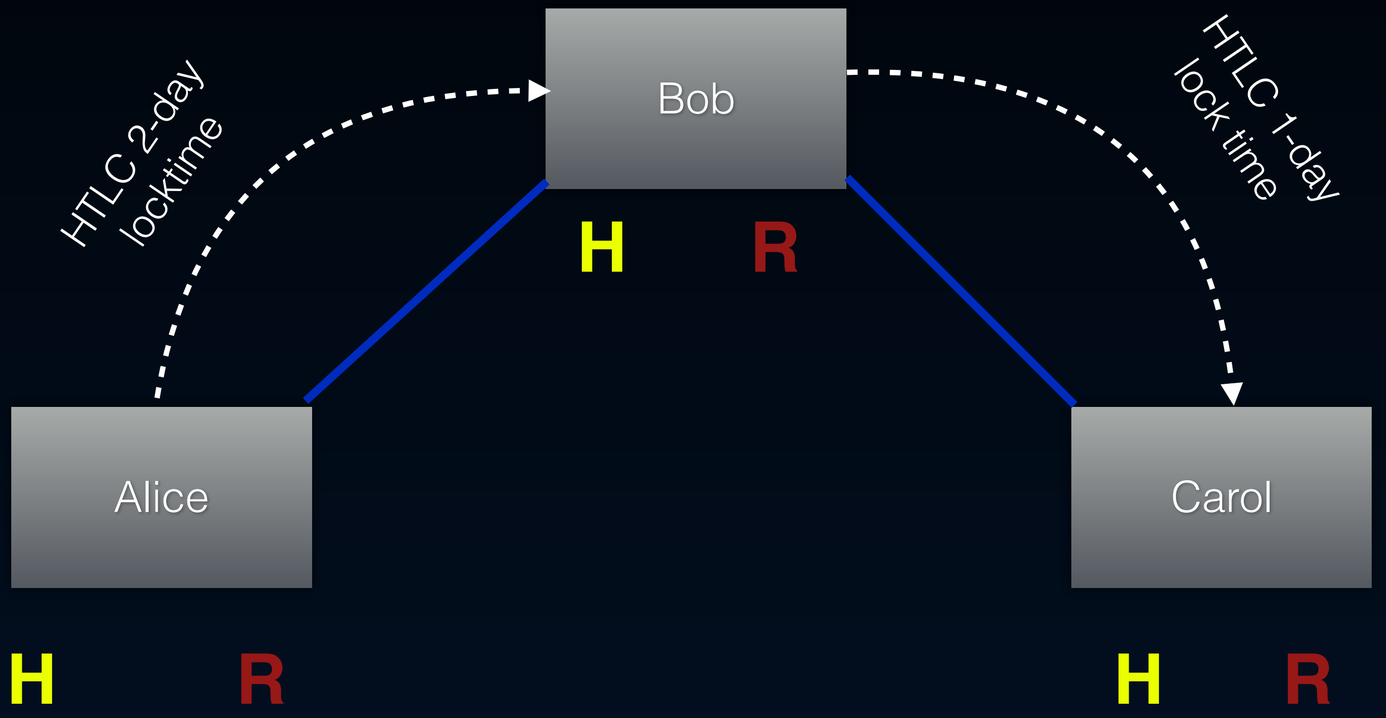
Recap onchain swap

- Needs an out-of-band communication channel (i.e. Alice and Bob need to communicate outside the protocol (to find each other, for negotiation, updates, etc…)
- All transactions happen on-chain (and need time for confirmation)
- Need to pay appropriate fees on both chains, and set reasonable refund timeouts
- Needs 2 TX (on both chains)

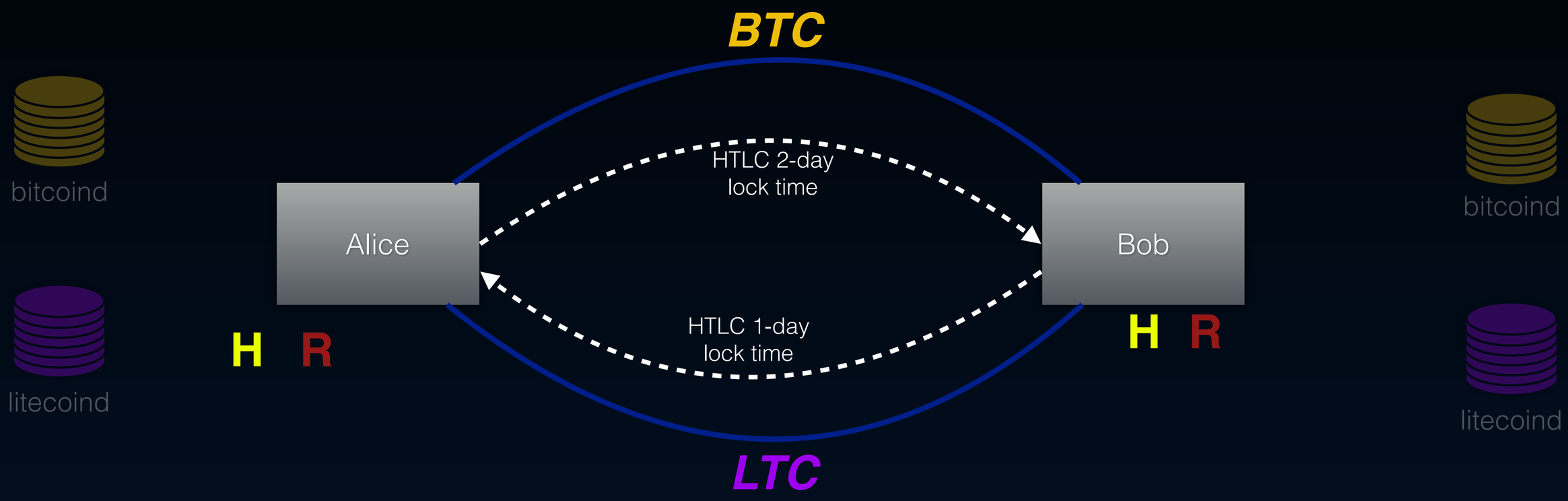
Lightning

- Now, we have lightning.. Yay! 😀⚡⚡⚡
- A lightning payment consists of a series of TXs happening atomically over a route of channels
- Very similar type of transactions (also HTLCs, slightly different for channel maintenance, but same base principle)

Recap: One-hop-Lightning payment example



The same but adapted for Swap (Alice is both, sender *and* receiver)



Status Lightning swaps

- the Lightning Paper mentions “Cross-Chain Payments” as a further possible use case for the Lightning Network, but no further details yet

- The lightning protocol specifications (BOLTs) also don’t mention swap related messages (yet)

- ***Conner Fromknecht*** from Lightning Labs did a proof-of-concept and performed a BTC/LTC swap on testnet using a modified “lnd” lightning daemon

- See blog article: https://blog.lightning.engineering/announcement/2017/11/16/ln-swap.html (proof-of-concept branch on GitHub and a howto, howto2 to recreate his demo)

Details proof of concept

modified “lnd” daemon (lnd already supports Bitcoin and Litecoin, but not simultaneously yet)

- added new custom cli-commands / parameters:
  - **`--ticker`** (to specify currency)
  - **`queryswaproutes`** (outputs a cross-chain route, used as input for sendtoroute, to explicitly send the payment over this route)Exchange rate statically configured
- exchange rate statically configured
- Basically, swap works like this:

```
lncli … queryswaproutes --in_amt=100000 --in_ticker=LTC --out_ticker=BTC
lncli … addinvoice --value=100000 --ticker=LTC    —> payment_hash
lncli queryswaproutes … | lncli sendtoroute --payment_hash <hash from
invoice>
```

Summary

- on-chain swaps already possible (and tooling getting better)
- Lightning supports cross-chain swaps perfectly by principle, it also seems like a natural fit for lightning. But not fully there yet.

  There still is the need for concrete specifications to be defined (gossip protocol, exchange rate publishing, advertising of supported currency swaps, etc..)

References

- "*Example 5: Trading across chains", 2011*, https://en.bitcoin.it/wiki/Contract#Example_5:_Trading_across_chains
- "*Alt chains and atomic transfers*", 2013, https://bitcointalk.org/index.php?topic=193281.msg2003765#msg2003765
- "*Malleability, deposits and atomic cross chain transfers*", 2014, https://bitcointalk.org/index.php?topic=515370.0
- "*ACCT using CLTV - More Effective than a sleeping pill!*", 2016 https://bitcointalk.org/index.php?topic=1340621.0
- BIP 0199: "*Hashed Time-Locked Contract transactions*"
- "BarterDEX – A Practical Native DEX", 2016, https://github.com/KomodoPlatform/KomodoPlatform/wiki/BarterDEX-%E2%80%93-A-Practical-Native-DEX
- "*Advisory: secret size attack on cross-chain hash lock smart contracts*", Dr. Mark B. Lundeburg, 2018, https://gist.github.com/markblundeberg/7a932c98179de2190049f5823907c016
- "Connecting Blockchains: Instant Cross-Chain Transactions On Lightning", 2017, https://blog.lightning.engineering/announcement/2017/11/16/ln-swap.html

Thank you for your time!

Find the slides here:

https://johannes.zweng.at/presentations/2018/atomic_swaps_BCHGraz/

Additional ref slides…

Bitcoin supported hashing OP_CODES

- OP_RIPEMD160
- OP_SHA1
- OP_SHA256
- OP_HASH160 *(SHA-256 + RIPEMD-160)*
- OP_SHA256 *(double SHA-256)*

Online Hash Function Demo

- SHA256: https://anders.com/blockchain/hash.html
- Multiple: https://www.fileformat.info/tool/hash.htm

Atomic Swap (2011 Hearn)

He proposed the following tx scheme
(before there was CLTV or CSV available)

- script:

```
IF
  2 <key A> <key B> 2 CHECKMULTISIGVERIFY
ELSE
  <key B> CHECKSIGVERIFY SHA256 <hash of secret A>
          EQUALVERIFY <hash of secret B> EQUALVERIFY
ENDIF
```

- spending data:

```
1 <sig A> <sig B>
```

or:

```
0 <sig B> <secret A> <secret B>
```

Funding TX, before CLTV

1) Alice creates the Funding TX but does NOT broadcast it

1 BTC

3) when Bob has signed the refund TX Alice broadcasts the locking TX and both wait until it confirms

"Funding" TX

This is 1 output, with 2 alternative spending conditions:

signature Alice

+

signature Bob

2) Alice creates this tx and let sign it by Bob, too

"refund Alice"
(nLocktime in future)

Going to Alice

OR

signature Bob

+

secret A + secret B

Bob can spend this output anywhere he wants as soon he knows secret A

(he knows already secret B as secret B is his own)

nLocktime vs. CLTV/CSV OP codes

nLockTime:

https://en.bitcoin.it/wiki/Protocol_documentation#tx and https://en.bitcoin.it/wiki/NLockTime

nLockTime is a field in the tx Header, tx cannot be included in a block before nLockTime hasn't been reached

If all inputs in a transaction have nSequence equal to UINT_MAX, then nLockTime is ignored

Locktime-Checking in Script:

BIP 65 - OP_CHECKLOCKTIMEVERIFY (allows a transaction output to be made unspendable until some point in the future - absolute time, in blocks or timestamp), 2014 (see using CTLV)

BIP 68 (Relative lock-time using consensus-enforced sequence numbers - 2015),

nSequence Number (32 bits field, Bit (1 << 22) defines type (time vs block), bit (1 << 31) is disable flag (if set, nSequence has no consensus-related meaning). Time is 512sec units, only 16 bits

Output cannot be used as inout in any tx until output has reached the defined age

BIP 112 later added OP_Code CHECKSEQUENCEVERIFY, to use nSequence as condition in script