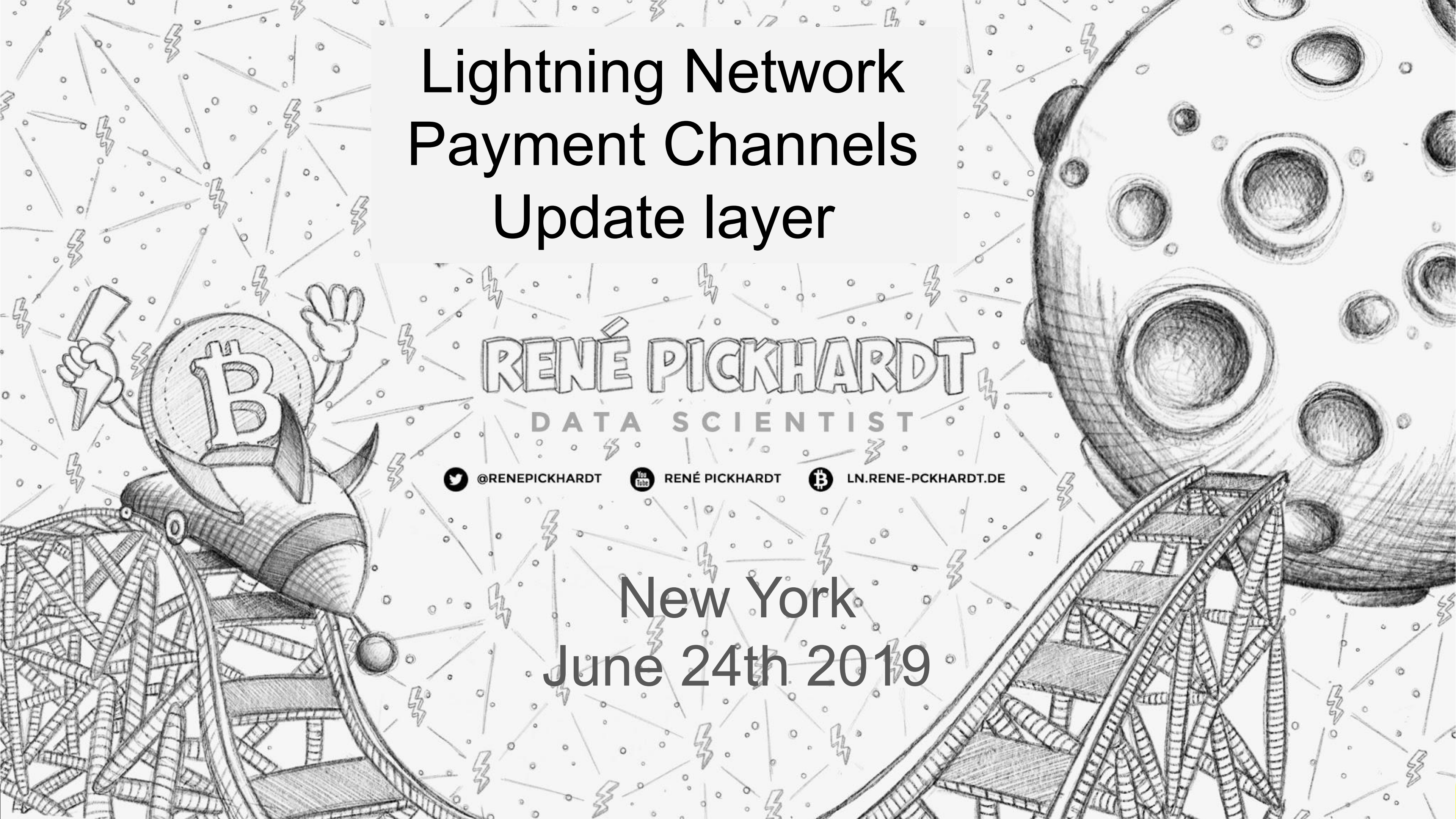
Lightning Network Payment Channels Update layer



New York
June 24th 2019

Disclaimer

- Please note the copyright notice at the end of the slide deck
- The pdf version - which you probably read has some weird formatting problems.
  - https://docs.google.com/presentation/d/1-eyceLlSmcLpbPJLzj6_CnVYQdo1AUP3y5XD716U-Lg has the source files without formatting problems and with animations
- This slidedeck is crowd funded (see last slide) if you wish to learn more or contribute
- I do not take any guarantee that the information in this slide deck are 100% correct.
  - Sometimes I made simplifications: which I point out
  - Also I could just have misunderstood something or just made a mistake

Outline

- Construction of payment channels in Bitcoin
    - A motivation from http
    - Channel construction
- Peer Protocol for channel Management (without HTLCs)
    - Peer messages for opening a channel
    - A word on Segwit and transaction malleability
    - 3 ways to close a channel

Construction of Payment channels (RSMCs)

(Chapter 1)

Purpose of a trustless bidirectional payment channel

- Sending payments between two partners
    - Instantly
        - Like updating a balance sheet
        - Only bound by network traffic over the internet
    - No direct involvement of the blockchain (as long as everyone plays by the rules)
- No need for any partner to trust the other side
    - The bitcoin network (and the blockchain) needs to be trusted
    - The bitcoin network will settle conflicts if they arise
- No fees or overhead when sending payments within the channel
- Payments can go in either direction back and forth
- Scale usage of Bitcoin
    - transfer of value can be achieved without hitting the blockchain

→ How can this technically be possible?

A comparison with HTTP

- HTTP is a Request Response Protocol by design
  - You (the client / Browser) can make a request
  - The server gives you a response
- According to the protocol a server cannot initiate communication with a client

- How can a website (e.g. Twitter) give us a notification that some new data is there?
  - By abusing the protocol
  - aka: Server push, long polling,
  - Took us about 10 years to figure that out

How does a HTTP server push work?

- You load a web page
- Javascript (AJAX) initiates an HTTP request
- Server looks if he has some information for the client
- If yes → response
- If no → defer answering to that request to some time in the future
  - If new information for the client is available
  - If client opens a new page and can't receive that response anymore
  - Just as a heartbeat mechanism to ask the client to make a new request
- From an end user perspective the server has initiated a conversation
  - The end user is usually not aware of the fact that an AJAX Request is outstanding

More details at: https://en.wikipedia.org/wiki/Push_technology

Payment channels are constructed by deferring the publication or broadcast of some TXs to the future

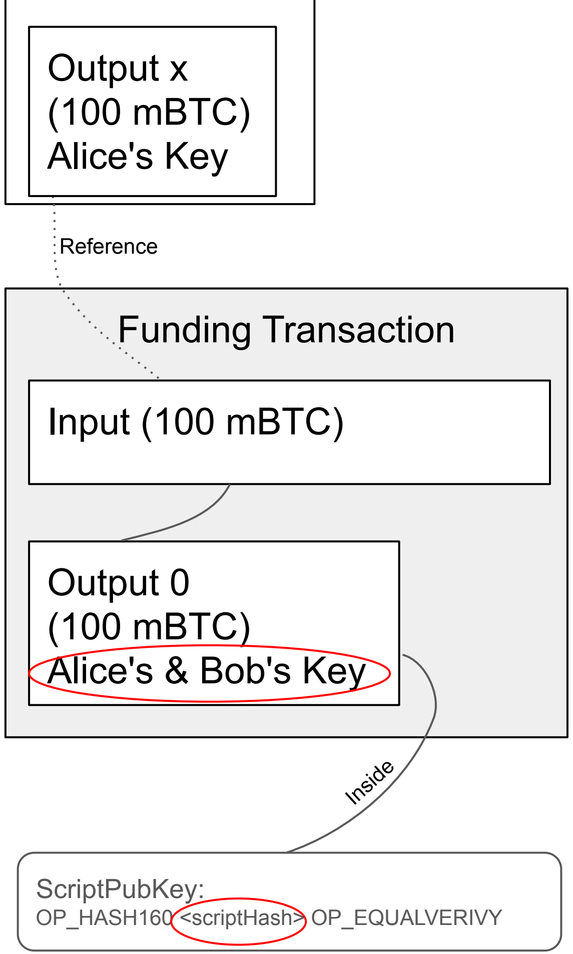
- Bitcoin is not a request response protocol
    - Rather a broadcast / gossip protocol
- What exactly is being deferred to the future?
    - Transactions spending outputs are held back from being broadcasted
    - Later Transactions are purposely tried to be double spent
        - Obviously only one of the transactions can be actually spent
        - Lightning is about making sure it is the correct one
    - We look at this in the next couple slides
- Payment channels are to Bitcoin what HTTP Server Push is to HTTP
    - Just a non obvious use of the Bitcoin protocol
    - It took us a little while to figure this feature of Bitcoin out (as for HTTP)

A payment channel is a 2-2 multisig Address together with some (smart?) contract

- A 2-2 multisig Address means that the output script of a transaction requires 2 keys in order to be able to spend the transaction
- It is a form of Pay to Script Hash
  - scriptPubKey (OUTPUT): OP_HASH160 <scriptHash> OP_EQUAL
  - scriptSig (INPUT): ...<signatures>... <serialized script>
- Case of a m-of-n multi-signature
  - The OUTPUT script (scriptPubKey) as above
    - OP_HASH160 <scriptHash> OP_EQUAL
  - The INPUT Script to spend the scriptPubKey looks like this:
    - scriptSig: 0 <sig1>...<serialized script>
    - Script: OP_m <pubKey1> … OP_n OP_CHECKMULTISIG

The smart contract (RSMC)

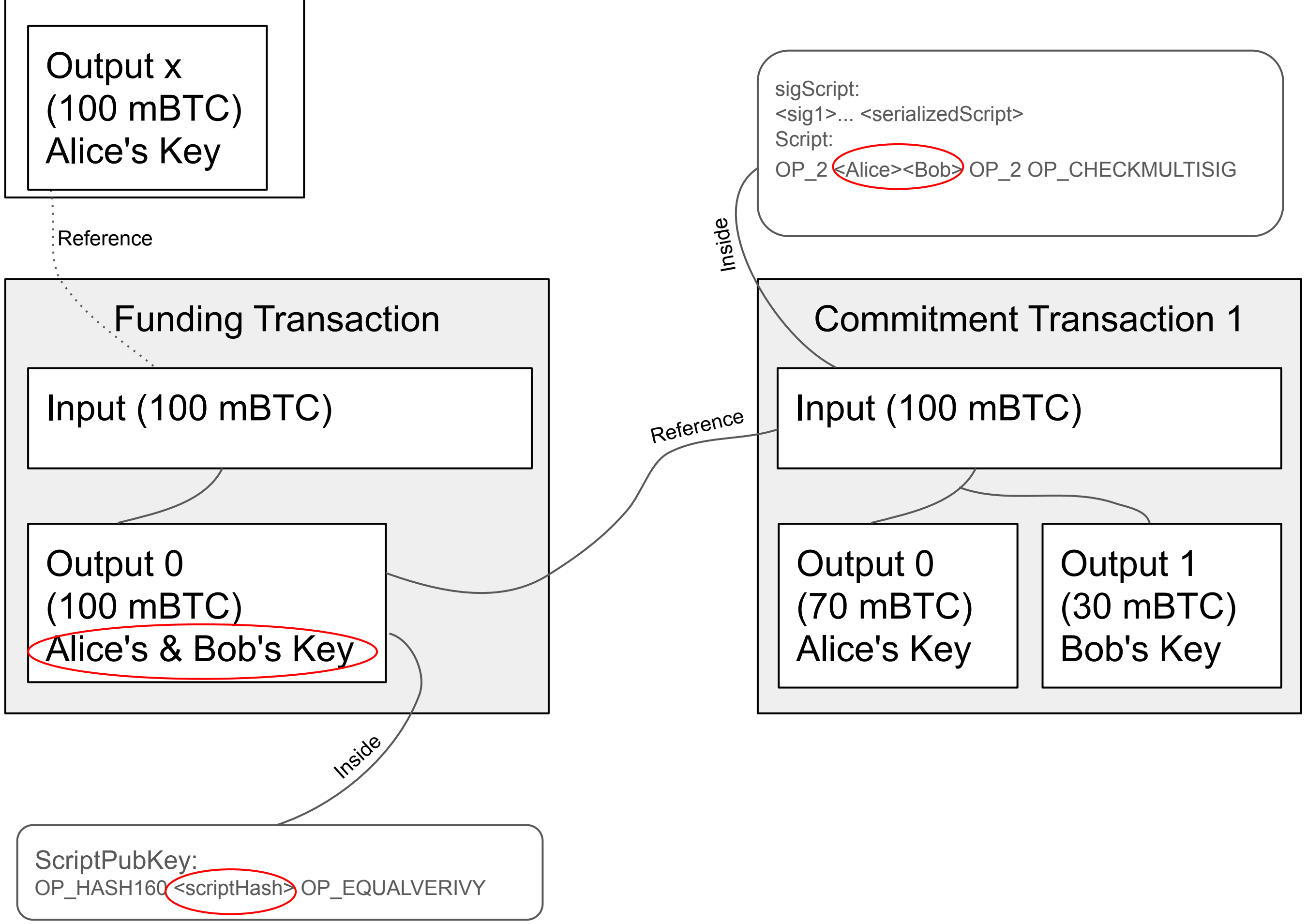
- The 2-2 multisig wallet has some funds
  - Seen on chain
- Who owns these funds?
  - Channel partners do not share the 2 private keys
  - Depends on how the funds are being spent
- The Transaction spending from the 2-2 Wallet
  - Will be kept secret (if possible)
  - Will encode the balance
  - Is called a commitment transaction (has to be signed BEFORE publishing funding tx)
- Technical details
  - Will be heavily attempted to be double spent (which by the bitcoin protocol is impossible)
  - Secures funds with timelocks
  - Allows for conditional hashed time locked outputs (used in routing)
- Revocable sequence maturity contract



UNPUBLISHED

Legend for colors:

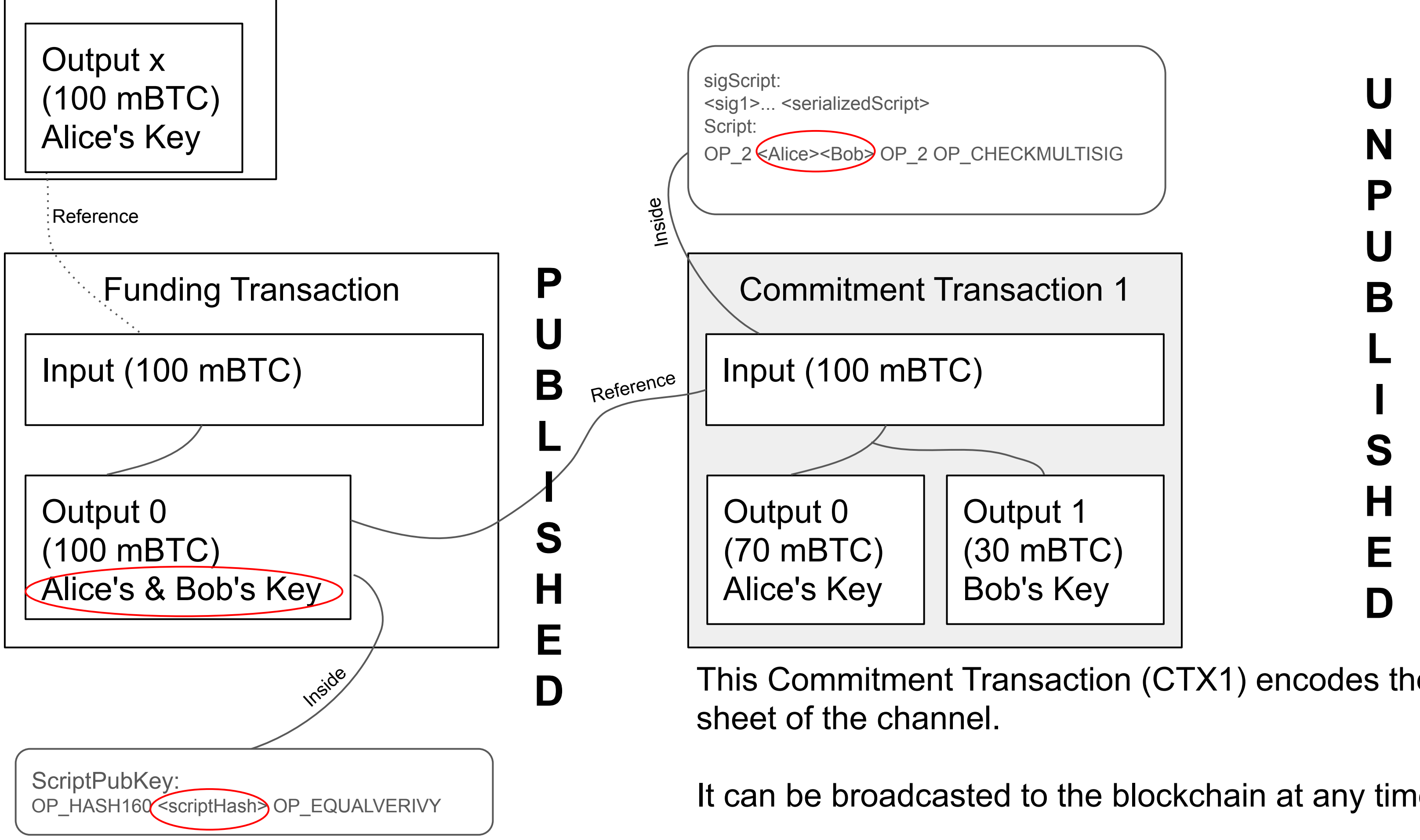
⬜	Broadcasted and mined
▨	Not broadcasted
🟩	Should be broadcasted
🟥	Should not be broadcasted
🟥	Cannot by minded (Consumed outputs are already spent)



ScriptPubKey:
OP_HASH160 <scriptHash> OP_EQUALVERIVY

sigScript:
<sig1>... <serializedScript>
Script:
OP_2 <Alice><Bob> OP_2 OP_CHECKMULTISIG

UNPUBLISHED

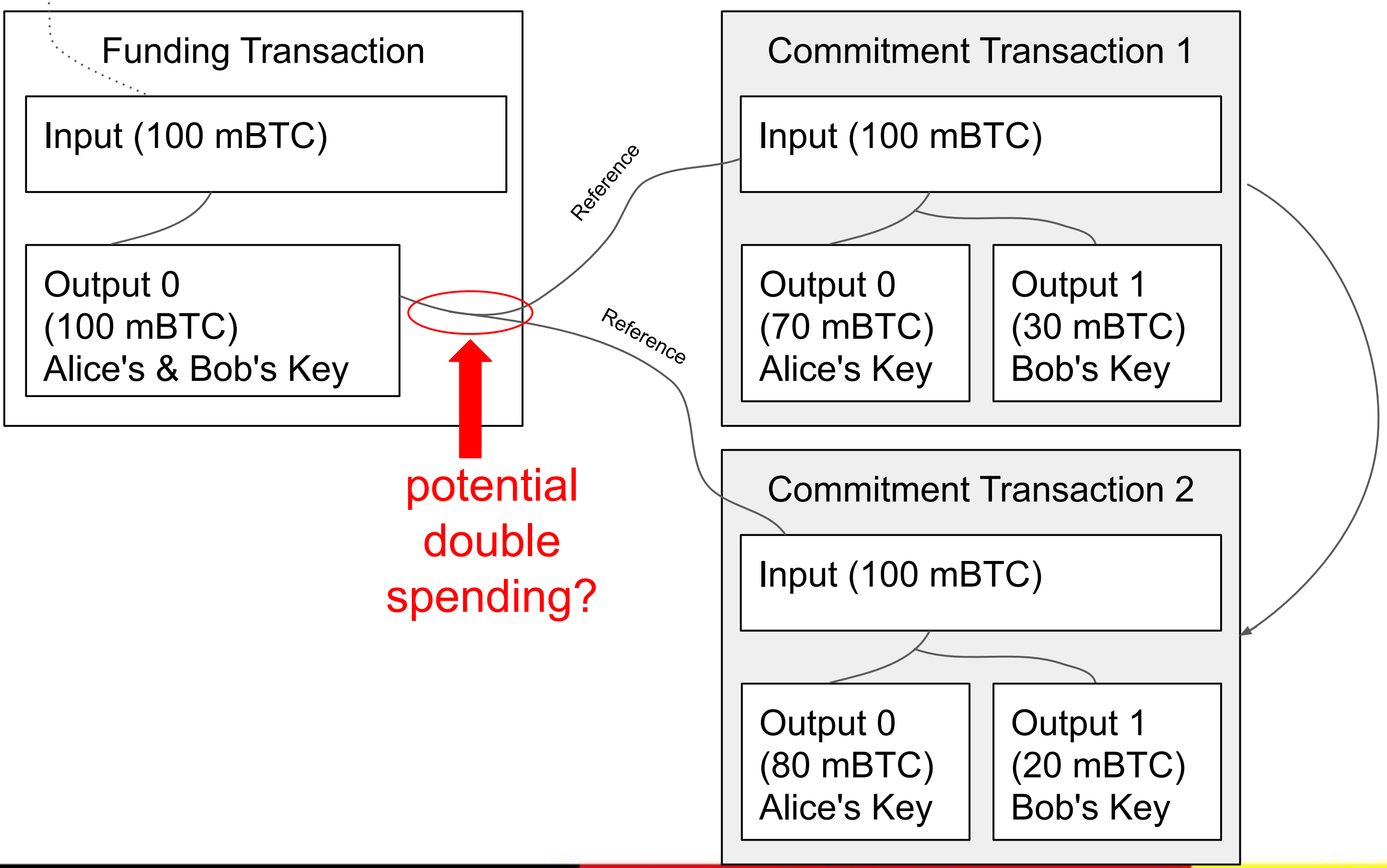


This Commitment Transaction (CTX1) encodes the balance sheet of the channel.

It can be broadcasted to the blockchain at any time

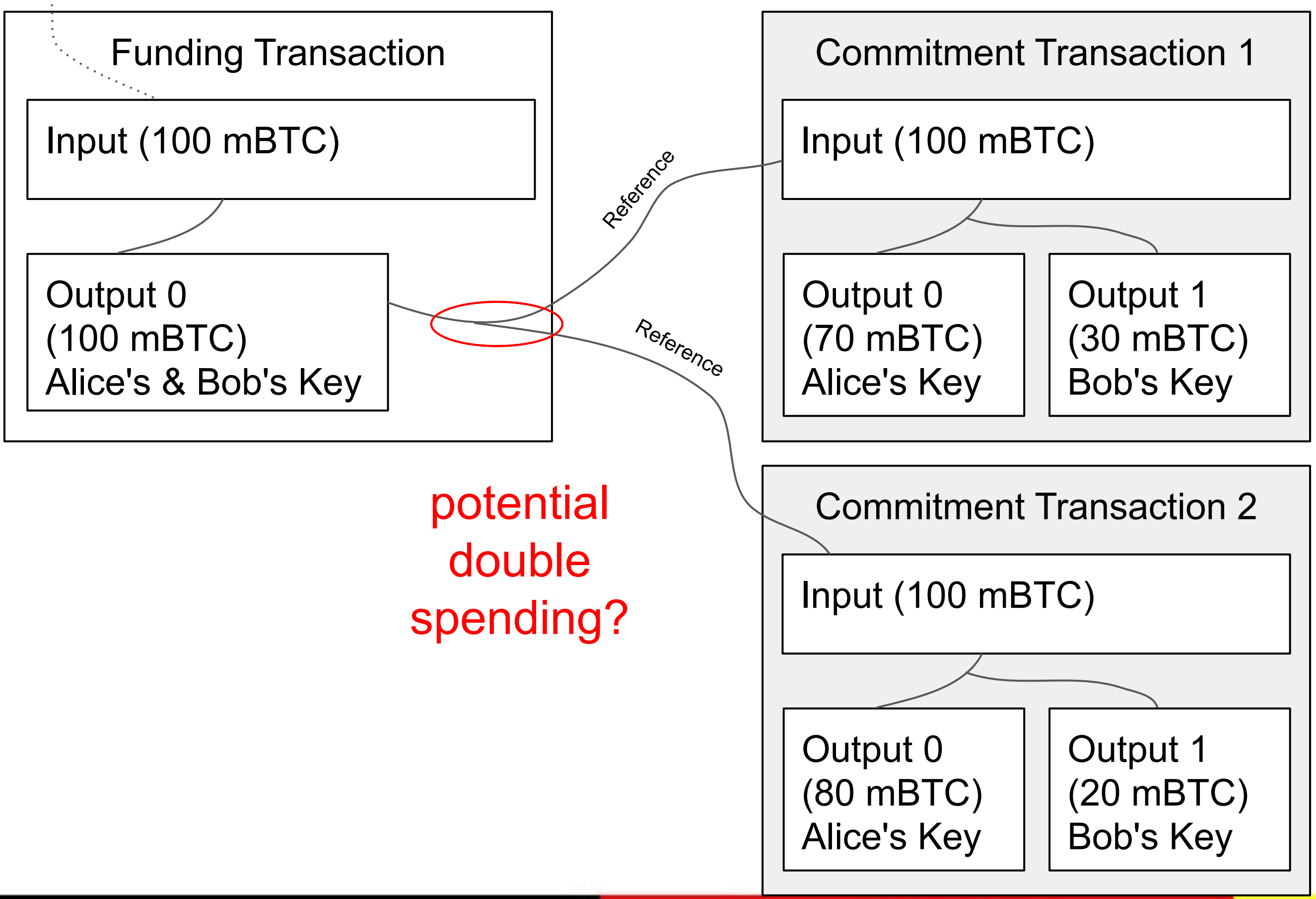
Funding Tx must use segWit outputs to prevent malleability

With CTX2 Bob updates the channel Balance

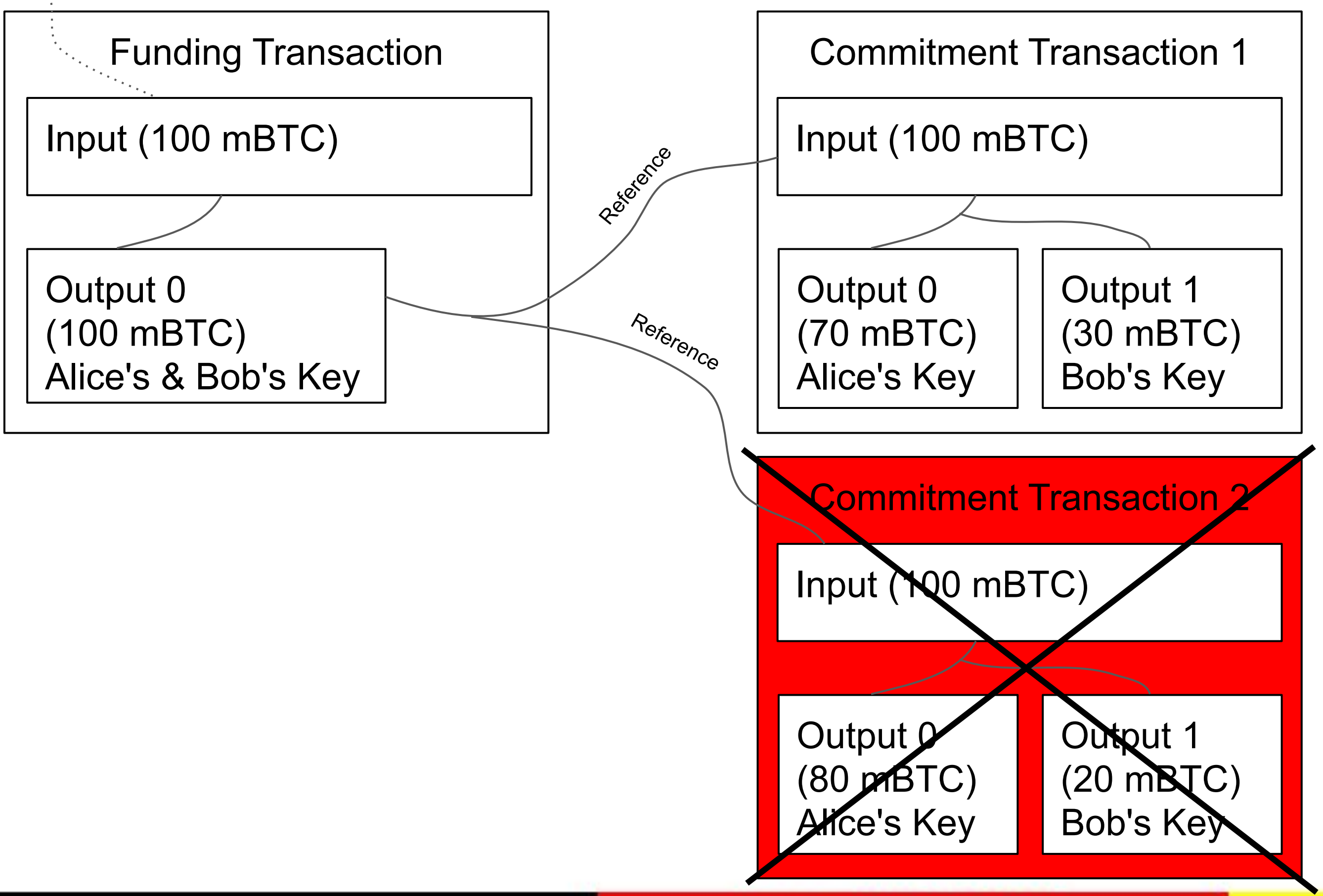


Bob effectively sends 10 mBTC to Alice

Anybody seeing a problem with this?

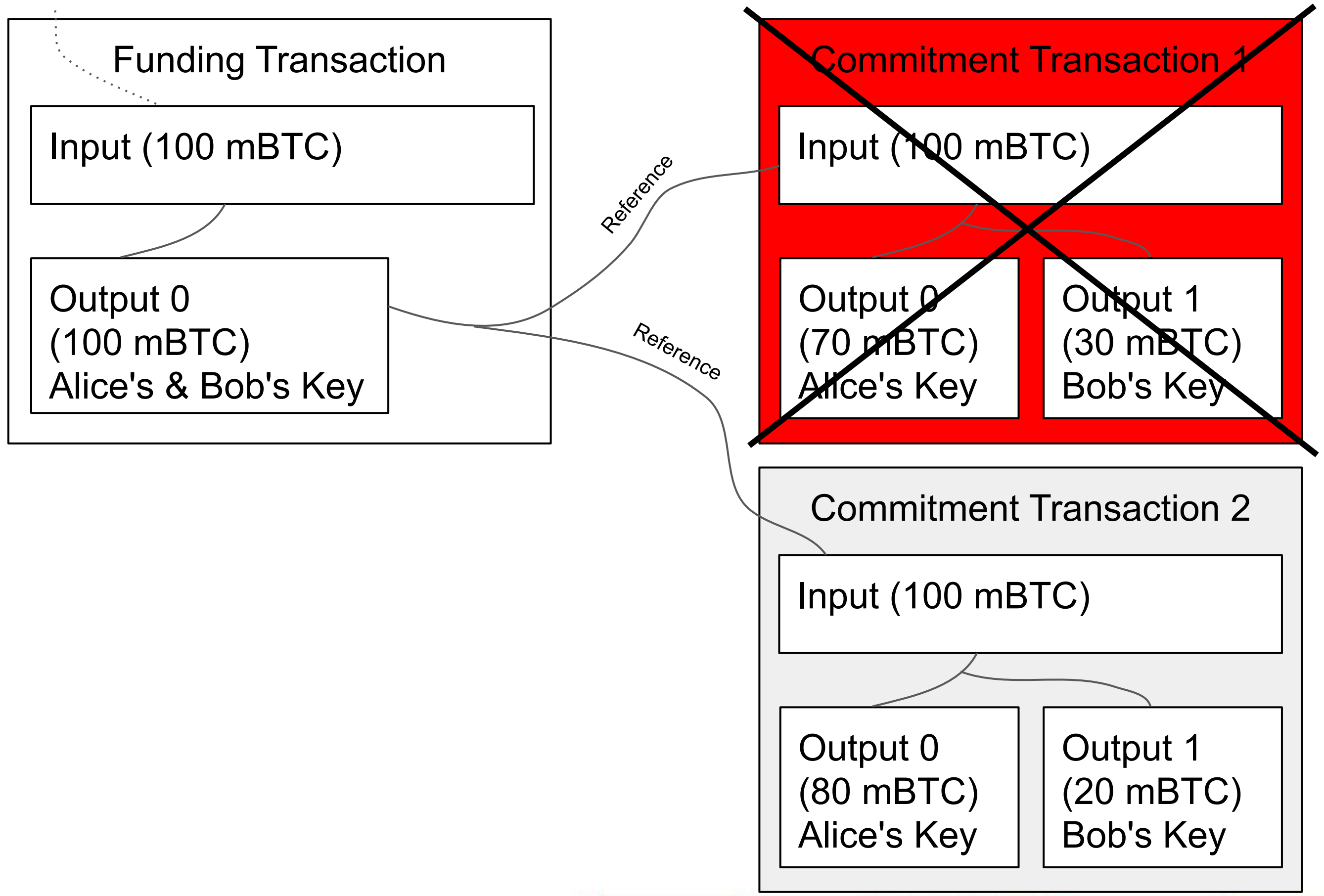


Bob broadcasts CTX1 which is successfully mined



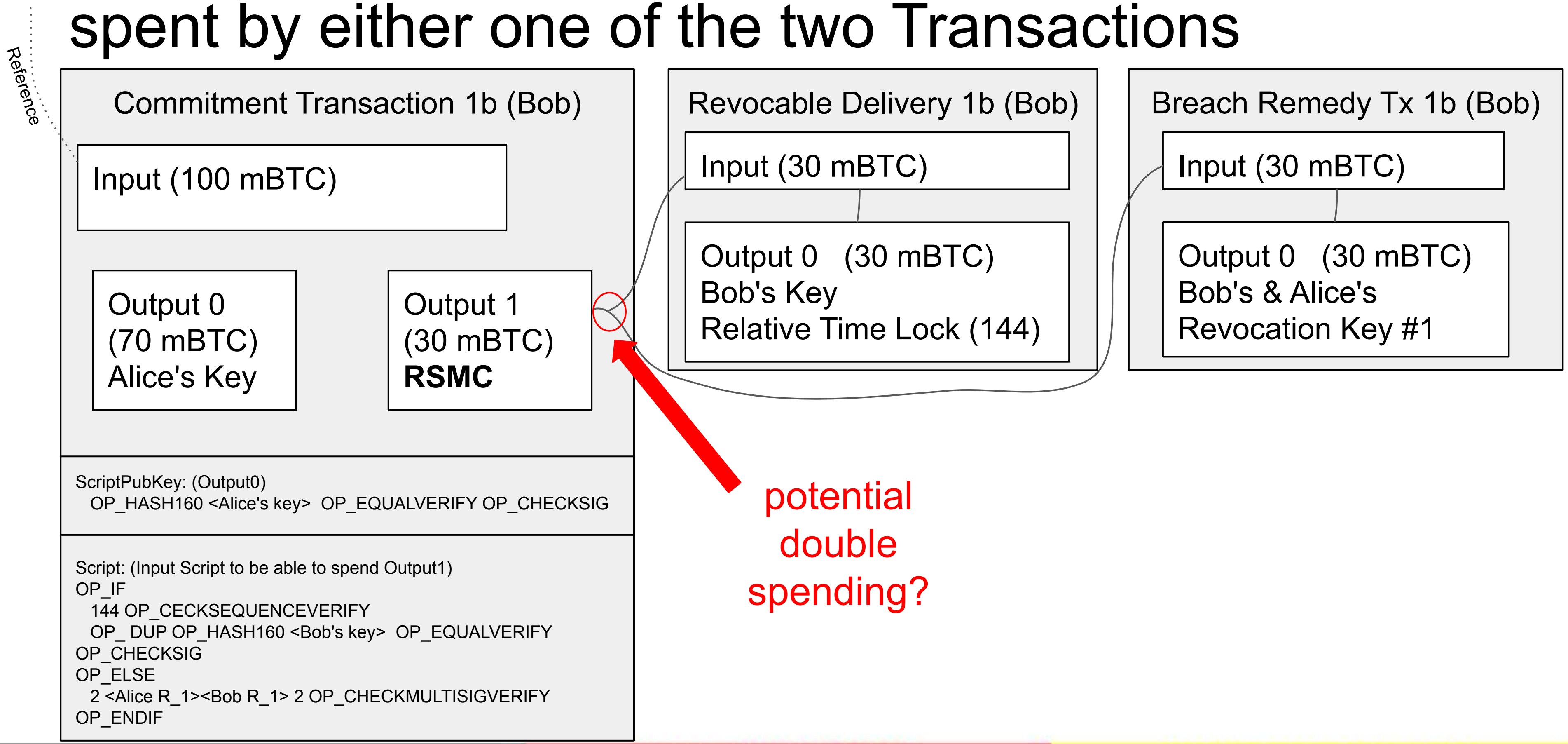
Bob just effectively stole 10 mBTC from Alice

Goal: Make old CTXs somehow unpublishable



- Modify the output of the CTX
- Fraudulent publishing will punish the publisher
- For example by giving the silent party the opportunity to claim all outputs of the CTX

We modify the output scripts of CTXs that it can be spent by either one of the two Transactions



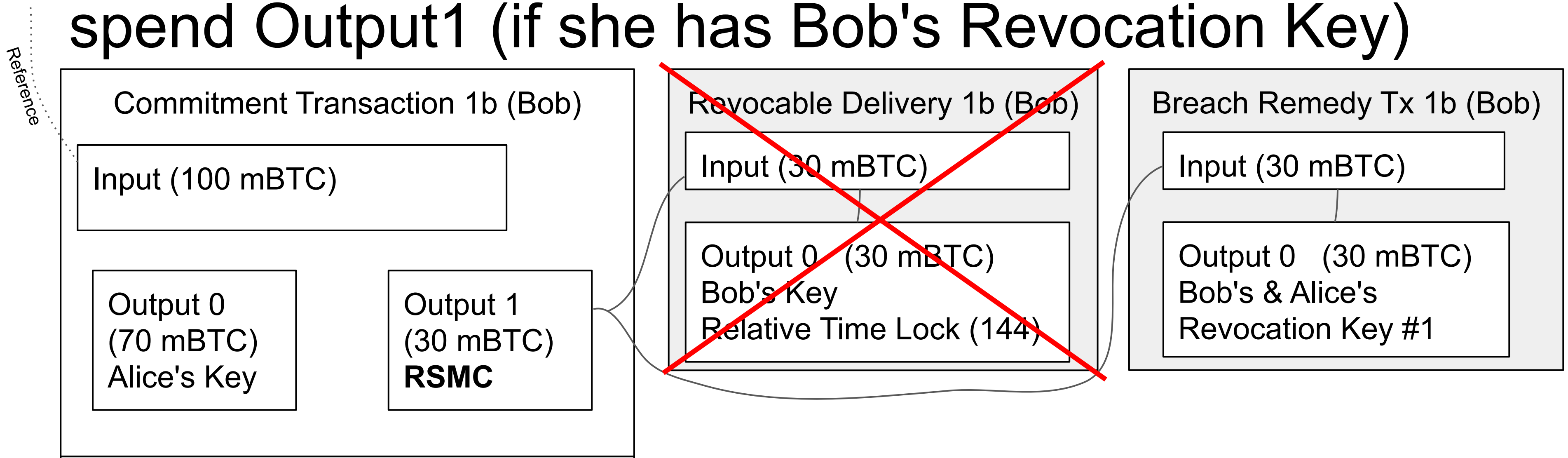
ScriptPubKey: (Output0)

```
  OP_HASH160 <Alice's key>  OP_EQUALVERIFY OP_CHECKSIG
```

Script: (Input Script to be able to spend Output1)

```
OP_IF
  144 OP_CECKSEQUENCEVERIFY
  OP_ DUP OP_HASH160 <Bob's key>  OP_EQUALVERIFY
OP_CHECKSIG
OP_ELSE
  2 <Alice R_1><Bob R_1> 2 OP_CHECKMULTISIGVERIFY
OP_ENDIF
```

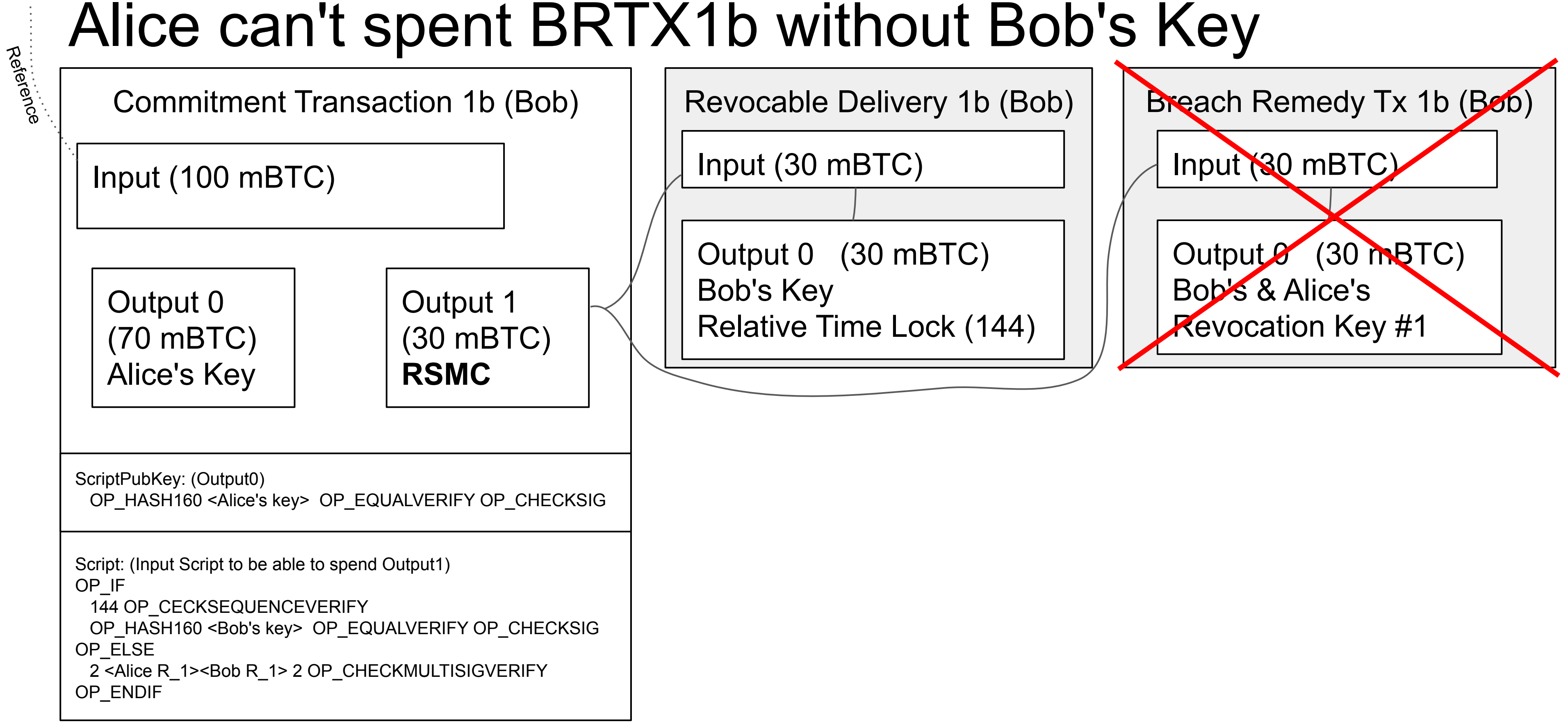
Within 144 Blocks after CTX1b is mined only Alice can spend Output1 (if she has Bob's Revocation Key)


Reference
Commitment Transaction 1b (Bob)
Input (100 mBTC)
Output 0 (70 mBTC) Alice's Key
Output 1 (30 mBTC) **RSMC**
~~Revocable Delivery 1b (Bob)~~
~~Input (30 mBTC)~~
~~Output 0 (30 mBTC) Bob's Key Relative Time Lock (144)~~
Breach Remedy Tx 1b (Bob)
Input (30 mBTC)
Output 0 (30 mBTC) Bob's & Alice's Revocation Key #1

```
ScriptPubKey: (Output0)
  OP_HASH160 <Alice's key>  OP_EQUALVERIFY OP_CHECKSIG
```

```
Script: (Input Script to be able to spend Output1)
OP_IF
  144 OP_CECKSEQUENCEVERIFY
  OP_HASH160 <Bob's key>  OP_EQUALVERIFY OP_CHECKSIG
OP_ELSE
  2 <Alice R_1><Bob R_1> 2 OP_CHECKMULTISIGVERIFY
OP_ENDIF
```

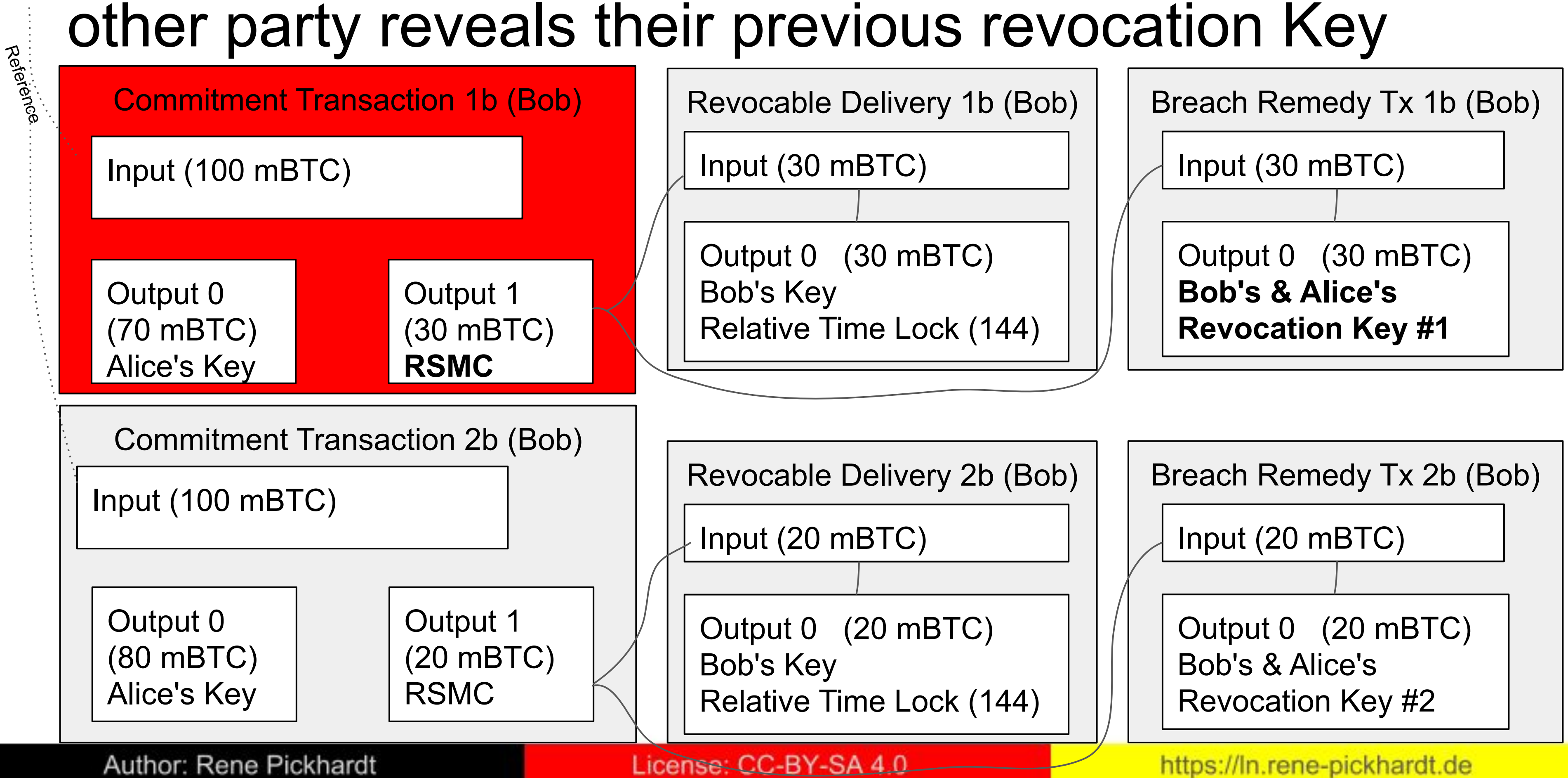
After 144 Blocks Bob can redeem his 10 mBTC & Alice can't spent BRTX1b without Bob's Key



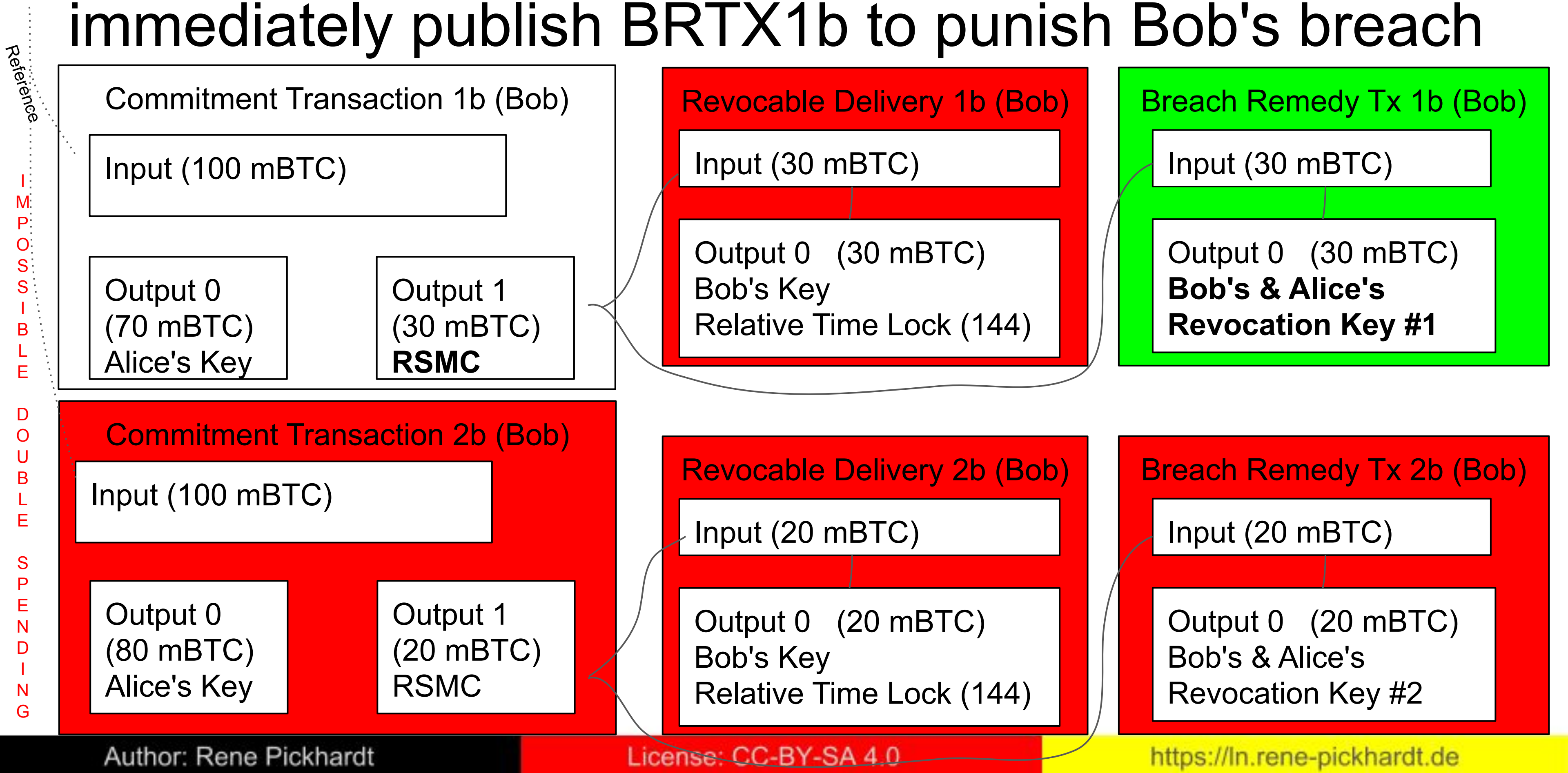
```
ScriptPubKey: (Output0)
  OP_HASH160 <Alice's key>  OP_EQUALVERIFY OP_CHECKSIG
```

```
Script: (Input Script to be able to spend Output1)
OP_IF
  144 OP_CECKSEQUENCEVERIFY
  OP_HASH160 <Bob's key>  OP_EQUALVERIFY OP_CHECKSIG
OP_ELSE
  2 <Alice R_1><Bob R_1> 2 OP_CHECKMULTISIGVERIFY
OP_ENDIF
```

A new Commitment Transaction is only signed if the other party reveals their previous revocation Key



Assume CTX1b is published and mined. Alice should immediately publish BRTX1b to punish Bob's breach


Reference
IMPOSSIBLE DOUBLE SPENDING
Commitment Transaction 1b (Bob)
Input (100 mBTC)
Output 0 (70 mBTC) Alice's Key
Output 1 (30 mBTC) **RSMC**
Revocable Delivery 1b (Bob)
Input (30 mBTC)
Output 0 (30 mBTC) Bob's Key Relative Time Lock (144)
Breach Remedy Tx 1b (Bob)
Input (30 mBTC)
Output 0 (30 mBTC) **Bob's & Alice's Revocation Key #1**
Commitment Transaction 2b (Bob)
Input (100 mBTC)
Output 0 (80 mBTC) Alice's Key
Output 1 (20 mBTC) RSMC
Revocable Delivery 2b (Bob)
Input (20 mBTC)
Output 0 (20 mBTC) Bob's Key Relative Time Lock (144)
Breach Remedy Tx 2b (Bob)
Input (20 mBTC)
Output 0 (20 mBTC) Bob's & Alice's Revocation Key #2

Some remarks on presented simplifications

- In order to be able to ascribe blame both sides get their own CTXs
  - Commitment Tx 1b, 2b, 3b,... (for Bob)
  - Commitment Tx 1a, 2a, 3a,... (for Alice)
- Transactions for every channel update
  - The Revocation Deliver TXs and the Breach Remedy TXs are adapted accordingly
- For every Update of the CTXs new Revocation Keys have to be created
  - Otherwise revealing the key once would not help for future updates
    - Future Breach Remedy Transactions could be spent directly
  - They are created from a Hierarchical Deterministic Wallet
- The Commitment Transactions will have even more outputs to support HTLCs
  - HTLC outputs will be needed for routing third party payments through one's channel
- Reading suggestion for a payment channel construction with less Overhead
  - eltoo: A Simple Layer2 Protocol for Bitcoin
    - Christian Decker, Rusty Russell (Blockstream)
    - Olaoluwa Osuntokun (Lightning Labs)
    - Summary at https://www.rene-pickhardt.de/thoughts-about-eltoo-another-protocol-for-payment-channel-management-in-the-lightning-network/

Some remarks on dust outputs

- In Bitcoin outputs MUST have a certain amount otherwise
  - the fees will be higher to spend the output than the amount attached to it
  - the blockchain can be spammed
- Dust limit depends on the script and is a few hundred satoshis.
  - C.f.: https://bitcoin.stackexchange.com/questions/10986/what-is-meant-by-bitcoin-dust
- If outputs in the commitment transaction are below the dust limit
  - they will be omitted
  - added to the fees
- While lightning supports tiniest payments those amounts cannot always be enforced on chain
- Making tiny payments / routing tiny amounts might make you lose those
  - Lighting implementations can be configured to not accept payments under a certain threshold

Standard to_local witness script for commitment txs

```
OP_IF
    # Penalty transaction
    <revocationpubkey>
OP_ELSE
    `to_self_delay`
    OP_CSV
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

Peer Protocol

(BOLT 02)

Purpose of the Peer Protocol

- Establish communication between two peers that are connected
- Assumes a working authenticated and encrypted Transport Layer
  - (on top of a single TCP socket)
- How will two nodes be able to maintain a RSMC (and HTLCs)?
- How will onion packages be sent between peers?
- Channel Management and operation of the channel
  - Channel establishment
  - Channel close
  - Normal Operation (omitted in this presentation but included to the routing talk)
    - Sending, accepting and forwarding payments
    - Updating fees
  - Message Retransmission (as communication transports are unreliable)

Format of Lightning messages

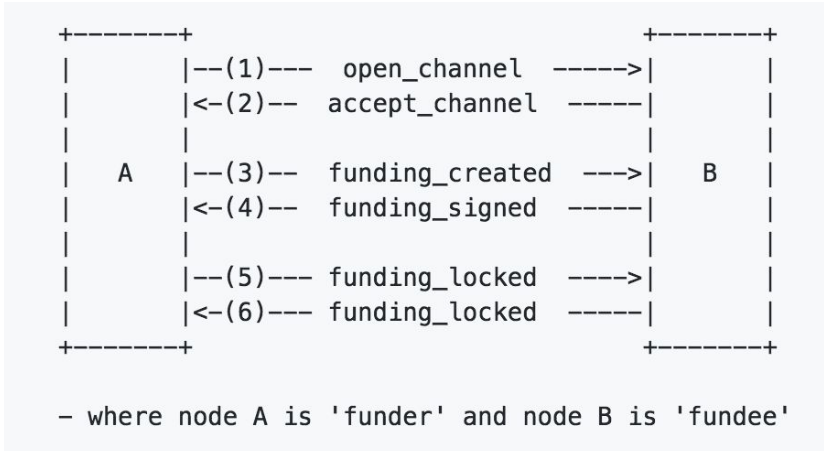
- 2 byte type field (big endian)
- Variable length payload
  - Max: 65535 Byte
- Format of the payload confirms to the type
- Even typed messages MUST be specified and MUST NOT be sent otherwise
- It's ok to be odd
- Logical groups of types
  - 0 - 31 Setup & Control
  - 32-127 Channel establishment and closure
  - 128-255 channel operation including setup and settlement of htlcs
  - 256-511 gossip protocol

Establishing a channel

(Chapter 6a / BOLT 02)

Establishing a channel

- Initializing and authenticating a peer connection
- 5 messages
  - open_channel
    - Suggests a channel with certain parameters
  - accept_channel
    - Aggrees to the channel
  - funding_created
    - Sends its signature for first commitment tx
  - funding_signed
    - Sends 2nd sig for commitment tx
  - funding_locked
    - Sends next_per_commitment_point (if funding tx has enough confirmations)



```
+-------+                              +-------+
|       |--(1)---  open_channel  ----->|       |
|       |<-(2)--  accept_channel  -----|       |
|       |                              |       |
|   A   |--(3)--  funding_created  --->|   B   |
|       |<-(4)--  funding_signed  -----|       |
|       |                              |       |
|       |--(5)--- funding_locked  ---->|       |
|       |<-(6)--- funding_locked  -----|       |
+-------+                              +-------+

- where node A is 'funder' and node B is 'fundee'
```

The open_channel message (type 32) ---->

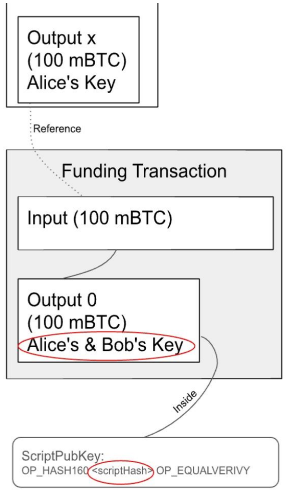
- **32: chain_hash**
- 32: temporary_channel_id
- 8: **funding_satoshis**
- **8: push_msat**
- 8: dust_limit_satoshis
- 8: max_htlc_value_in_flight_msat
- 8: htlc_minimum_msat
- 2: to_self_delay
- 2: max_accepted_htlcs
- 33: funding_pubkey
- Basepoints (as in BOLT 03 for privacy reason with watchtowers)
  - 33: Revocation
  - 33: Payment
  - 33: Delayed_payment
  - 33: Htlc
  - 33: first_per _commitment
- **1: channel_flags**, 2: shutdown_len, shutdown_scriptpubkeyv

The accept_channel message (type 33) <----

- 32: temporary_channel_id
- 8: dust_limit_satoshis
- 8: max_htlc_value_in_flight_msat
- 8: channel_reserve_satoshis
- 8: htlc_minimum_msat
- **4: minimum_depth**
- 2: to_self_delay
- 2: max_accepted_htlcs
- 33: funding_pubkey
- Basepoints (as in BOLT 03 for privacy reason with watchtowers)
  - 33: Revocation
  - 33: Payment
  - 33: Delayed_payment
  - 33: Htlc
  - 33: first_per _commitment
- 2: shutdown_len, shutdown_scriptpubkey

The funding_created message (type 34) ---->

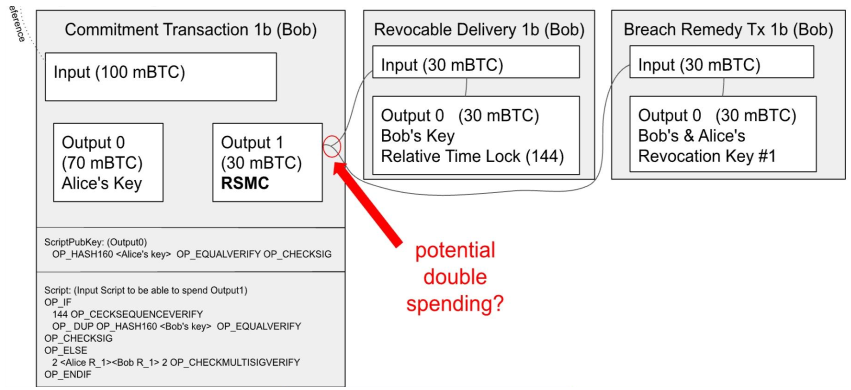
- 32: temporary_channel_id
  - MUST be same as in open channel message
- 32: funding_txid
  - Tx has to be non malleable and MUST NOT be broadcasted at that time
- 2: funding_output_index
- 64: signature
  - Valid signature using the funding_pubkey for the initial commitment tx
  - If incorrect channel MUST be failed



U
N
P
U
B
L
I
S
H
E
D

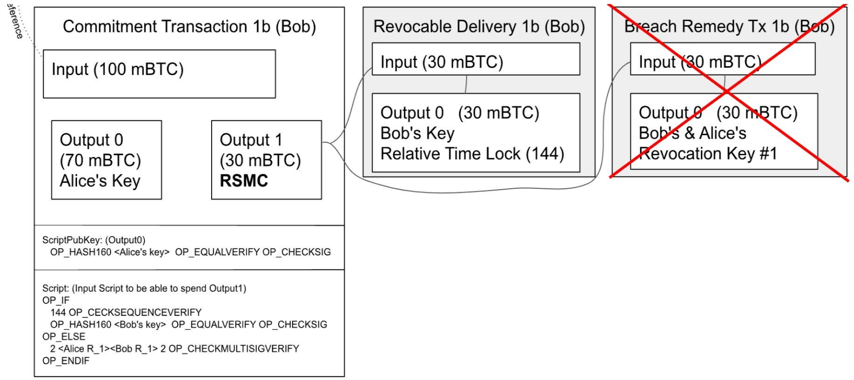
The funding_signed message <----

- 32: channel_id
  - Funding_txid ^ funding_output_index
- 64: signature
  - 2nd signature for the first commitment tx
  - If invalid
    - fail channel
    - MUST NOT broadcast the funding_tx
  - If valid SHOULD broadcast the funding_tx



```
ScriptPubKey: (Output0)
  OP_HASH160 <Alice's key>  OP_EQUALVERIFY OP_CHECKSIG
```

```
Script: (Input Script to be able to spend Output1)
OP_IF
  144 OP_CECKSEQUENCEVERIFY
  OP_ DUP OP_HASH160 <Bob's key>  OP_EQUALVERIFY
OP_CHECKSIG
OP_ELSE
  2 <Alice R_1><Bob R_1> 2 OP_CHECKMULTISIGVERIFY
OP_ENDIF
```

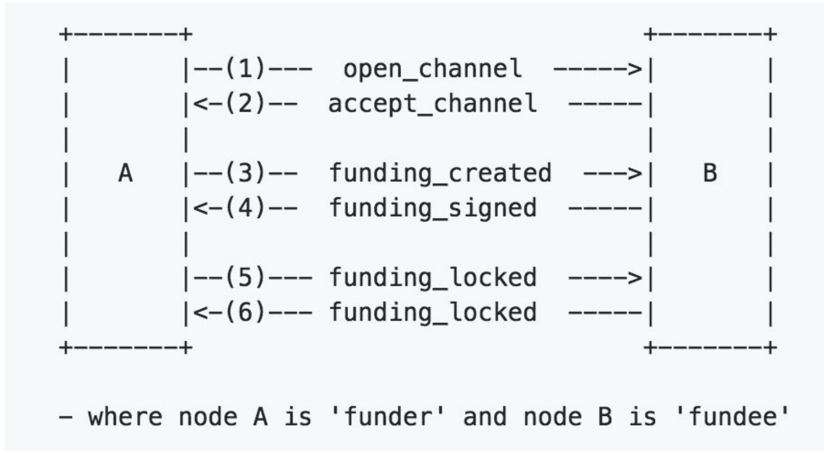


```
ScriptPubKey: (Output0)
  OP_HASH160 <Alice's key>  OP_EQUALVERIFY OP_CHECKSIG
```

```
Script: (Input Script to be able to spend Output1)
OP_IF
  144 OP_CECKSEQUENCEVERIFY
  OP_HASH160 <Bob's key>  OP_EQUALVERIFY OP_CHECKSIG
OP_ELSE
  2 <Alice R_1><Bob R_1> 2 OP_CHECKMULTISIGVERIFY
OP_ENDIF
```

On Transaction Malleability and the need for segWit

- After signatures for commitment tx are exchanged via funding_signed
    - Funding tx is published by A
    - B can catch this funding tx
- If B malleates the tx it gets a new txid
- Ways to malleate the tx via
    - Signature
    - sigScript
    - https://en.bitcoin.it/wiki/Transaction_malleability
- B can try to publish the malleated version
- Commitment tx refers to txid
- If txid of funding tx is malleated the commitment tx becomes worthless
- B can now blackmail A
    - A cannot access the funds A provided without the help of B
    - Since B did not invest some funds this way for blackmail is "cost free" for B
- Segwit mitigates the possibility to malleate a transaction

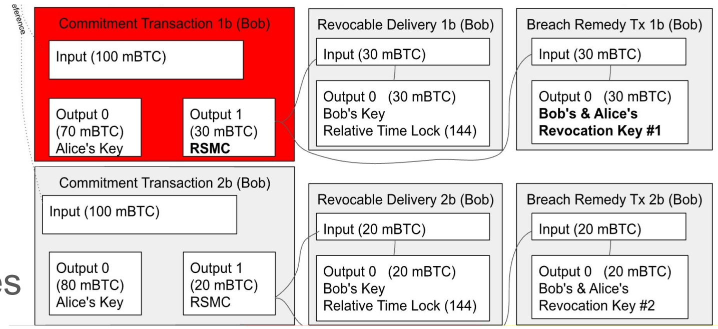


```
+-------+                              +-------+
|       |--(1)---  open_channel  ----->|       |
|       |<-(2)--  accept_channel  -----|       |
|       |                              |       |
|   A   |--(3)--  funding_created  --->|   B   |
|       |<-(4)--  funding_signed  -----|       |
|       |                              |       |
|       |--(5)--- funding_locked  ---->|       |
|       |<-(6)--- funding_locked  -----|       |
+-------+                              +-------+

- where node A is 'funder' and node B is 'fundee'
```

The funding_locked message (type 36) ----> <----

- 32: channel_id
- 33: next_per_commitment_point
  - As defined in BOLT 03 Key derivation
  - Used for next commitment transaction
    - Future base points cannot be derived from old ones
      - This shall improve the privacy
      - Especially when using watching services



Commitment Transaction 1b (Bob)
Input (100 mBTC)
Output 0 (70 mBTC) Alice's Key
Output 1 (30 mBTC) RSMC
Revocable Delivery 1b (Bob)
Input (30 mBTC)
Output 0 (30 mBTC) Bob's Key Relative Time Lock (144)
Breach Remedy Tx 1b (Bob)
Input (30 mBTC)
Output 0 (30 mBTC) Bob's & Alice's Revocation Key #1
Commitment Transaction 2b (Bob)
Input (100 mBTC)
Output 0 (80 mBTC) Alice's Key
Output 1 (20 mBTC) RSMC
Revocable Delivery 2b (Bob)
Input (20 mBTC)
Output 0 (20 mBTC) Bob's Key Relative Time Lock (144)
Breach Remedy Tx 2b (Bob)
Input (20 mBTC)
Output 0 (20 mBTC) Bob's & Alice's Revocation Key #2

- Funding_locked message is necessary for the channel to become operational
  - Sender MUST wait until minimum_depth of funding tx before sending this message
  - Recipient SHOULD forget the channel if message does not come
    - Mitigates DoS attack risks.

Closing a channel

(Chapter 6b / BOLT 02 / BOLT 05)

3 ways for closing a channel (BOLT 05)

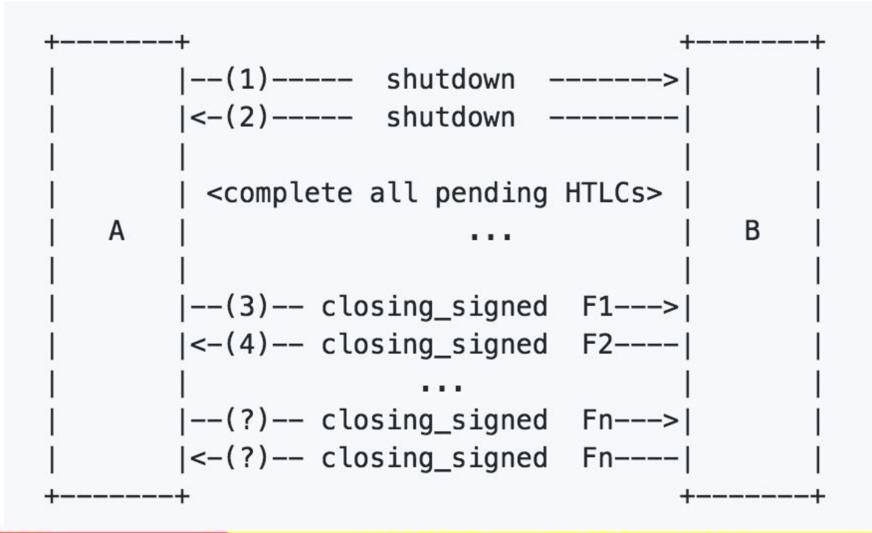
- The good (Mutual close)
  - 1 closing transaction
  - Similar to commitment tx but no pending payments
  - Part of BOLT 02 - peer protocol
- The bad (unilateral close)
  - The protocol was breached
    - Look out for "... MUST fail the channel" in the BOLTs
    - Could be accidentally e.g. hardware failure
  - One side publishes latest commitment transaction
  - Part of BOLT 05 - onchain
- The ugly (revoked transaction close)
  - One party (deliberately?) tries to cheat
  - Software bugs
  - Part of BOLT 05 - onchain

The good way - Mutual Close (always preferable)

- Excludes the time lock of the funds
  - Partners can spend the funds directly

- Fee can be negotiated when closing takes place
  - Potentially saves fee

- 2 Messages are included
  - shutdown
    - To signal intend
  - closing_signed
    - Mainly to negotiate fees



```
+-------+                              +-------+
|       |--(1)-----  shutdown  ------->|       |
|       |<-(2)-----  shutdown  --------|       |
|       |                              |       |
|       | <complete all pending HTLCs> |       |
|   A   |                 ...          |   B   |
|       |                              |       |
|       |--(3)-- closing_signed  F1--->|       |
|       |<-(4)-- closing_signed  F2----|       |
|       |              ...             |       |
|       |--(?)-- closing_signed  Fn--->|       |
|       |<-(?)-- closing_signed  Fn----|       |
+-------+                              +-------+
```

The shutdown message

- Type: 38
- Data
  - 32: channel_id
  - 2: len
  - len: scriptpubkey (one of the following script templates)
    - `P2PKH:` `OP_DUP OP_HASH160 20` 20-bytes `OP_EQUALVERIFY OP_CHECKSIG`
    - `P2SH:` `OP_HASH160 20` 20-bytes `OP_EQUAL`
    - `P2WPKH:` `OP_0 20` 20-bytes
    - `P2WSH:` `OP_0 32` 32-bytes

- MUST NOT be before funding_created / funding_signed messages
- MAY be done before funding_locked messages
- No future add_update_htlc messages will be accepted
- No future update messages will be accepted

The closing_signed message

- Type: 39
- Data
    - 32: channel_id
    - 8: fee_satoshis
    - 64: signature

- All htlcs must be settled / cleared in the current Commitment Tx
- If fees of one roundtrip of closing_signed message are equal
    - SHOULD sign and broadcast the closing tx
    - MAY close the connection
- Fee_satoshis should converge
    - MUST propose a value strictly between received fee_satoshis and previously-sent one.
- No real risk for DOS as channel partner could fail the channel

The bad way - Unilateral / Force Close

- The most recent commitment transaction is pushed to the chain
- Channel stops immediately to be operational
- Almost exact channel state is seen on chain
    - Balance
    - All fully committed htlcs in flight whose outputs are higher than dust
        - Including preimages
- Spend htlcs with either
    - Htlc - success transactions (assuming knowledge of the preimage)
    - Htlc - timeout transactions

```
OP_IF
    # Penalty transaction
    <revocationpubkey>
OP_ELSE
    `to_self_delay`
    OP_CSV
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

The ugly way - Revoked Transaction Close

- An old commitment transaction was published
- Revocation secrets have been shared via revoke_and_ack message
  - We see how in the following 'normal channel operation' part
- **Other side claims all funds within a time lock**
  - This is the penalty mechanism that incentivises both parties to be honest and not go for the ugly way
  - For the mechanism to work there should always be a channel reserve on channels so that there is actually some penalty to collect.
- A watching service can help to do so
- If no claiming with the time lock the breaching party can spend their outputs
- This case should never ocurr

What can be seen by channel closes on chain?

- Type of close
- Channel state
    - In case of force close
    - In mutual close only the final balance can be seen
- Pending htlc outputs
    - If not below dust limit
    - There might be a fight about htlc sizes round the dust limit
        - Nodes have an interest to be larger than the dust limit
        - Privacy aware people might want to be below the dust limit

Rational for Retransmission messages

- Communication transports are unreliable
  - Cannot be assumed updates_add_htlc were received unless commitment_signed was received
  - Only store updates upon receipt of commitment_signed
  - This ensures retransmission after channel_reestablishment

- Type: 136 (channel_reestablish)
- Data:
  - 32: channel_id
  - 8: next_local_commitment_number
  - 8: next_remote_revocation_number
  - 32: your_last_per_commitment_secret
  - 32: my_current_per_commitment_point

References and helpful links

- https://github.com/lightningnetwork/lightning-rfc
- http://noiseprotocol.org/noise.html
- https://www.youtube.com/user/RenePickhardt
- https://bitcoin.stackexchange.com/questions/tagged/lightning-network
- https://en.bitcoin.it/wiki/Script
- https://en.bitcoin.it/wiki/Transaction
- https://lightning.network/lightning-network-paper.pdf
- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- https://en.bitcoin.it/wiki/Transaction_malleability
- https://bitcoin.org/en/glossary/txid

Copyright notice

About this slide deck

The purpose is to help spreading education about the Lightning Network Protocol so that the technology will be adopted more quickly by more people. This shall be my contribution to the Bitcoin / Lightning Network Community.

This slide deck was presented during Chaincodelabs Lightning Residency program in June 2019. It is part of https://commons.wikimedia.org/wiki/File:Introduction_to_the_Lightning_Network_Protocol_and_the_Basics_of_Lightning_Technology_(BOLT_aka_Lightning-rfc).pdf. To the best of my knowledge the original file is the most comprehensive work making an introduction to the BOLT standard.

The slides are part of my effort to create a book about the lightning network. You can follow that effort at: https://github.com/renepickhardt/The-Lightning-Network-Book or you can support the effort at my fundrasing pages at: https://tallyco.in/s/lnbook or at: https://www.patreon.com/renepickhardt or at 1GZx8tWgDd21Rd8b1QdMrzdZGHgyfVkzaD part of this effort also consists of creating video tutorials and teaching materials on my Youtube Channel over at: https://www.youtube.com/user/RenePickhardt

This work was funded (sorted by amount of contribution from top to bottom) by: Me personally, fulmo.org, everyone who contributed to the above mentioned fundraiser and George Danzer.

Thank you to the lightning developers and people in various telegram groups for helpful discussions