Optimal Trustless One-way Payment Channel for Bitcoin

Thomas Shababi[1], Joël Gugger[2], and Daniel Lebrecht[1]

[1] DigiThink, Neuchâtel, Switzerland
info@digithink.ch
[2] HES-SO Master, Lausanne, Switzerland
joel.gugger@master.hes-so.ch

Abstract. The largest challenge in Bitcoin for the coming years is scalability. Currently, Bitcoin enforces a 1 Megabyte block-size limit which is equivalent to ~7 transactions per second on the network. This is not sufficient in comparison to big payment infrastructure such as credit card processors, which allows tens of thousands of transactions per second and even more in peaks like Christmas. To address this, some proposals modifying the transaction structure (like SegWit), some proposals modifying the block-size limit (such as SegWit2x) and others creating a second layer based on top of the Bitcoin protocol (like Lightning Network) exist. In the same idea of the Lightning Network, we propose a one-way payment channel that allows two parties to transact off-chain while minimizing the number of transactions needed in the blockchain in a secure and trustless way.

Keywords: Crypto-currencies, Bitcoin, Payment channels, State channels, Threshold ECDSA signatures

1 Introduction

Decentralized crypto-currencies such as Bitcoin [5] and its derivatives employs a special decentralized public append-only log based on proof-of-work called the *blockchain* to protect against equivocation in the form of *double-spending*, i.e., spending the same funds to different parties. In a decentralized crypto-currency, users transfer their funds by publishing digitally signed transactions. Transactions are confirmed only when they are included in the blockchain, which is generated by currency miners that solve proof-of-work puzzles. Although a malicious owner can sign over the same funds to multiple receivers through multiple transactions, eventually only one transaction will be approved and added to the publicly verifiable blockchain.

But the blockchain is slow and could be expensive in fees when comes the time to broadcast transactions. Scalability is one of the biggest challenge in blockchain systems these days, as mentionned before, some proposals are focused on the blockchain structure and modify the consensus, others like the Lightning Network [6] are focused on a second layer of transaction where transactions are

created off-chain and the blockchain itself is used as an conflict resolving system and a source of truth. These proposals are called payment channels and provide a wide number of advantages and possibilites.

1.1 Our contribution

Most of commecrial transactions are unidirectional and bidirectional channels are not necessary. Streaming payments in buying services context are mostly unidirectional and bidirectional channels implies both parties to policy the chain and listen the network, which greatly increase the complexity. In this paper we propose a simplified scheme aimed to be use in a context of a provider providing goods or services at many clients and this provider wants to receive the payments into payment channels for rapidity and convenience.

In our scheme the client is represented by Carol, she want to buy goods or services to the provider Bob. Bob is likely to belly serveral times goods or services to Carol and wants to receive payments into a channel to optimize his costs. For this scheme to be realistic, some requirements are assumed. The channel must stay open for undefined amount of time and the client must not be obliged to stay online and watch the blockchain to be safe, only the receiver, i.e., the provider, must stay online to be safe. The provider does not want to lock any fund for the clients, if he needs to send money to some clients, it is asumed that these transactions are regular on-chain transaction or via other channels. When clients send money into channels, the provider also must be able to manage when and how many funds are settle from which channels, so he must be able to settle a channel without closing it. A client, who has blocked funds specifically for the provider, must be able to, with the provider cooperation, withdraw an arbitrary amount out of the channel without closing it.

We propose two definitions to qualify a state channel and generalize the analysis of different implementation.

Definition 1. *A channel is trustless if the funds' safety for every players* $p_i \in \mathcal{P} = \{\mathcal{P}_0, \ldots, \mathcal{P}_n\}$ *at each steps* $\mathcal{S}$ *of the protocol does not depend on players'* $\Delta p = \mathcal{P} - p_i$ *behavior.*

Definition 2. *A channel is optimal if the number of transaction* $\mathcal{T}(\mathcal{C})$ *needed to claims the funds for a given constraint* $\mathcal{C}$ *is equal to the number of moves* $\mathcal{M}(\mathcal{C})$ *needed to satisfy the constraint at any time without breaking the first definition.*

For a containt $\mathcal{C}$ *in a channel* $\mathcal{P}_1 \rightarrow \mathcal{P}_2$*, refunding* $\mathcal{P}_1$ *where the amount* $M \geq 0$ *of* $\mathcal{P}_2$ *require* $\mathcal{M}(\mathcal{C}) \geq 2$ *because of the intermediary revokation's state. An optimal scheme require* $\mathcal{T}(\mathcal{C}) = \wedge\mathcal{M}(\mathcal{C}) = 2$.

The channel is trustless in the meaning of Def. 1 and is optimal in the meaning of Def. 2. This paper describe how to achive a trustless one-way payment channel for Bitcoin. Our work is inspired by the Lightning Network and "Yours Lightning Protocol" [1].

2 Building Blocks

In the following the concepts and sub-protocols used in this work are described in more detail.

2.1 Channel State

The channel state is expressed by two indexes i and n, hereinafter also $\texttt{Channel}_{i,n}$. Both indexes are independent and can only be positively incremented. Index i represents the offset of the multisig address where the channel's funds are locked. Index n represents the offset use to create the revocation secrets, this secret is used after in smart contracts.

A channel state always depends on an account a, this account is defined when the channel is created between the client and the server and never changes during his life. We need to share public hierachical deterministic addresses between the client and the server. Let's define the hierachical deterministic Bitcoin account path as:

$$\forall a \geq 2, \exists \texttt{xPriv}_a \mid \texttt{xPriv}_a = \texttt{m/44'/0'/a'}$$
$$\forall a \geq 2, \ \exists \texttt{xPub}_a \ \mid \texttt{xPub}_a = \texttt{m/44'/0'/a'}$$

For a given account a at $\texttt{Channel}_{i,n}$, the protocol and transactions depend on the private multi-signature node $\mathit{\Pi}$, the public multi-signature node π, the private revocation node Ω, the public revocation node ω, and the private secret node Θ. Let's define these nodes as:

$$\mathit{\Pi}_i = \texttt{xPriv}_a \texttt{ /0/i}$$
$$\pi_i \ = \texttt{xPub}_a \texttt{ /0/i}$$
$$\Omega_i = \texttt{xPriv}_a \texttt{ /1/i}$$
$$\omega_i \ = \texttt{xPub}_a \texttt{ /1/i}$$
$$\Theta_n = \texttt{xPriv}_a \texttt{ /2'/n'}$$

It is worth noting that $\mathit{\Pi}_i$, π_i, Ω_i, and ω_i aren't hardened derivations, instead of Θ_n. This because we need to be able to compute the public keys π_i and ω_i from the $\texttt{xPub}_a$.

Channel Dimensions The channel dimension, noted $|\texttt{Channel}|$, depends of the number of indexes present in the state. Let's define the channel dimension:

$$N = |\texttt{Channel}_{i,n}| = 2$$

Revocation Secret The revocation sercret $\mathit{\Phi}_{i,n}$ corresponds to the state $\texttt{Channel}_{i,n}$ and depends on the secret Θ_n and the revocation key Ω_i.

$$\mathit{\Phi}_{i,n} = \texttt{HMAC}(\Theta_n, \Omega_i)$$

The secret is the HMAC of Θ_n and Ω_i. Both indexes are used to protect Carol from the Old Settlement Attack With Weak Secret (see 4.3).

2.2 Smart Contracts

Two types of smart contract are used in the payment channel scheme. The fisrt one is a standard 2-out-of-2 multi-signature script and the second is a custom script used to prevent the client from broadcasting old transactions.

Multisig Contract The multi-signature contract at $\texttt{Channel}_{i,n}$, hereinafter $\texttt{Multisig}_i$, can be constructed with Carol's π_i key, and Bob's π_i key. Let's define the $\texttt{Multisig}_i$ script:

OP_2 <π_i^{carol}> <π_i^{bob}> OP_2 OP_CHECKMULTISIG

Revocable PubKey Contract Bob and Carol may wish make an output to Carol which Carol can spend after a timelock and Bob can revoke if it is an old state. The next contract, for a $\texttt{Channel}_{i,n}$, use Carol's ω_i key, Bob's ω_i key, and Carol's secret $\Phi_{i,n}$.

```
OP_IF
  <ω_i^carol> OP_CHECKSIG
  <timelock> OP_CHECKSEQUENCEVERIFY OP_DROP
OP_ELSE
  <ω_i^bob> OP_CHECKSIGVERIFY
  OP_HASH160 <Hash160(Φ_{i,n})> OP_EQUAL
OP_ENDIF
```

With this contract Carol can spend this output after the timelock with the script signature:

<Ω_i^{carol} signature> OP_TRUE

In the case if Carol broadcasts an older transaction Bob can revoke it with the script signature:

<Carol's $\Phi_{i,n}$> <Ω_i^{bob} signature> OP_FALSE

Bob has a head start during which, if he knows the secret $\Phi_{i,n}$ generated by Carol, he can spend the money while Carol cannot. This mechanism prevents Carol from broadcasting older transactions which do not match the current $\texttt{Channel}_{i,n}$.

2.3 Transactions

A transaction is noted $\texttt{Transaction}^{i,n}_{<>}$ to denote the name of the transaction, on which indexes this transaction depends—here on indexes i and n—and who has already signed this transaction—denoted by the <>. If a transaction is signed by Carol the transaction is noted $\texttt{Transaction}^{i,n}_{<carol>}$. Transactions that appear in blue on figures are only owned fully signed by Carol and red ones only by Bob, i.e., only the owner can broadcast the transaction.

Funding Transaction The funding transaction, hereinafter $\texttt{FundingTx}^{i}_{<>}$, is the transaction sendting funds to the first multisig address. This transaction depends only on the state index i used by the multisig contract and is fully signed as soon as Carol signs it.

A funding transaction is never broadcast by Carol before she owns the corresponding refund transaction that allows her to get her money back off the channel. This refund transaction has only one output that goes to the revocation contract. To be able to revoke this contract Bob has to know the secret $\Phi_{i,n}$, so if no transaction are made Bob cannot revoke the contract.

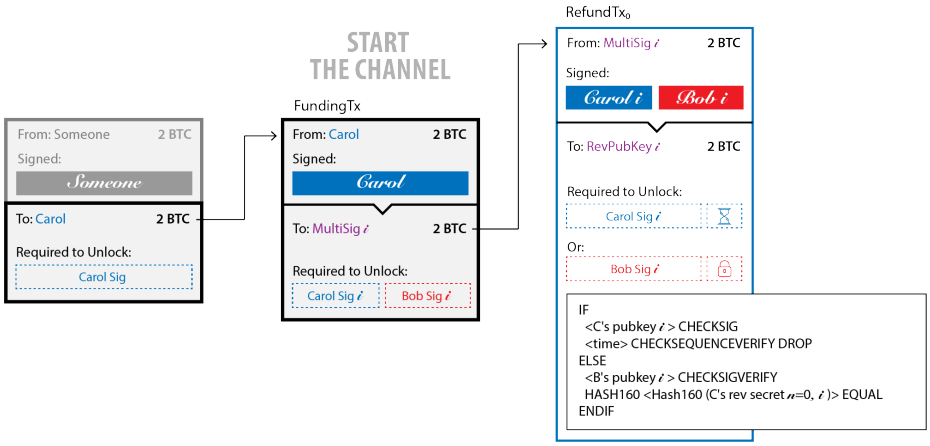

Fig. 1. Funding transaction that start the channel by sending money in the first multisig address with the first refund transaction that allows Carol to close the channel if no transaction is made.

Refund Transaction The refund transaction, hereinafter $\texttt{RefundTx}^{i,n}_{<>}$, is a transaction that keeps track the balances of Carol and Bob at $\texttt{Channel}_{i,n}$ and allows Carol to close the channel if Bob does not repsond or does not cooperate anymore. This transaction have one input or more—which come from the multisig address corresponding to the state index i—and two outputs. The first output represent the amount still owned by Carol, and the second—which can be non-present in the transaction if the balance is equal to zero—is the amount owned by Bob. This second output is not present when the channel is open and when Bob settle the channel.

The Carol balance is send to a revocation contract corresponding to the channel state. This prevent Carol to broadcast an old refund transaction such as $\texttt{RefundTx}^{i,n-1}_{<>}$. The amount owned by Bob is sent directly to Bob's address. The refund transaction is broadcasted by Carol so the fees are substracted to the first output, owned by Carol.

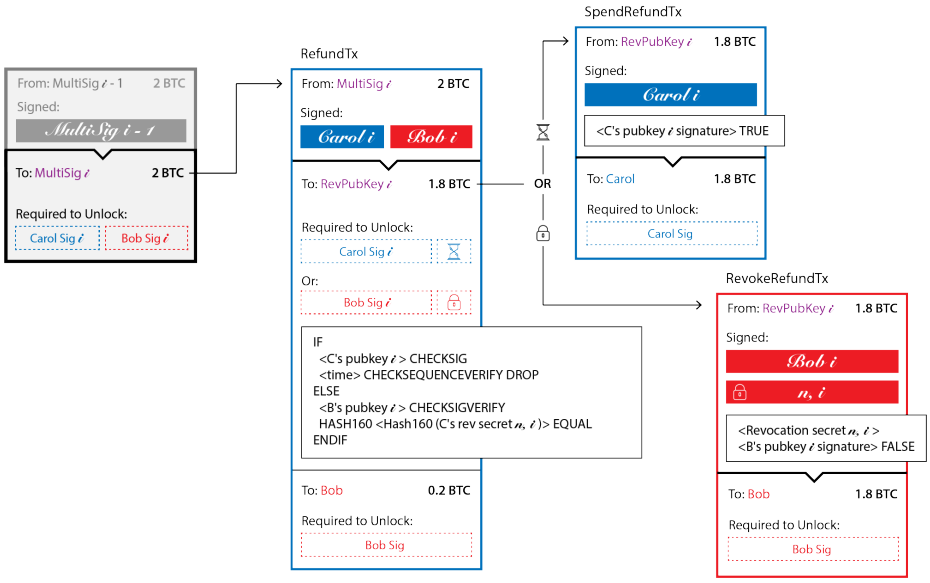

Fig. 2. Refund transaction based on the current multisig address with the associated spend and revoke transactions that allows Carol to get her money back and Bob to revoke the contract if he knows the secret.

Because a refund transaction spends funds from a multisig address, she must be signed by Carol and Bob to be considered fully signed. The revocation contract used in the output of Carol can be spent with a spend refund transaction after a timelock delay. She just needs to sign the output with her Ω_i key to unlock the funds. Bob can directly revoke the contract, without delay, if he knows the secret $\Phi_{i,n}$ and sign with his Ω_i key.

Settlement Transaction The settlement transaction, hereinafter also mentioned as $\texttt{SettlementTx}^{i,n}_{<>}$, is a transaction that keeps track the balances of Carol and Bob at $\texttt{Channel}_{i,n}$ and allows Bob to settle the channel without closing it. Because the settlement transaction spends the funds present into the multisig address, both Carol and Bob need to sign to consider the transaction as a fully signed transaction. Fees are substracted from Bob's owned output, because he is responsible to broadcast the transaction and settle the channel.

A settlement transaction has always one output that sends Bob's balance directly to Bob's address and one output that send the remaining funds to the next multisig address $\texttt{Channel}_{i+1,n}$. Because the funds are sends to the next multisig address a post settlement refund transaction is created—Carol needs a way to get her money back off the channel. This transaction has the same structure as the first refund transaction—one output to the next revocation contract—because the funds owned by Bob was already settle.

If Bob broadcast the fully signed settlement transaction, Carol has two choices (i) continue to transact on the channel with the new multisig address and (ii) close the channel with her post settlement transaction. It is worth noting that the secret for the revocation contract is $\Phi_{i+1,n}$, so in the case of a new transaction $\texttt{Channel}_{i+1,n}$ becomes $\texttt{Channel}_{i+1,n+1}$ and then, the secret for $\texttt{Channel}_{i,n-1}$ is shared:

$$\Phi_{i,n-1}(\texttt{Channel}_{i+1,n+1}) = \Phi_{i+1,n}$$

Post Settlement Refund Transaction The post settlement transaction aim to spend funds from the next multisig address directly to a revocation contract. As explained before, this contract is not revocable by Bob if no transaction is made after the settlement transaction, but when Carol sends an amount to Bob she shares the secret needed to revoke the contract, thus she cannot broadcast this transaction that is now attach to an old state.

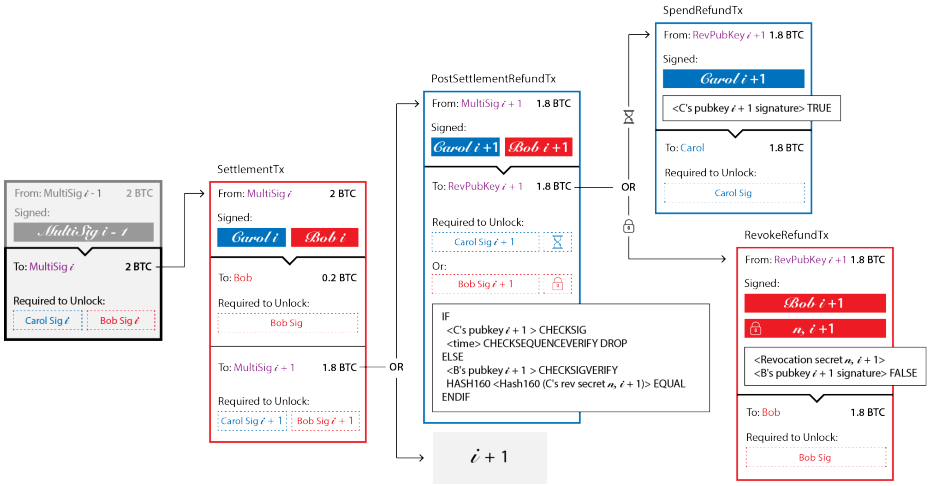

Fig. 3. Settlement transaction that allows Bob to settle the channel with moving the remaining funds to the next multisig address with the post settlement refund transaction that allows Carol to close the channel directly after the settlement.

Withdraw Transaction The withdraw transaction, hereinafter $\texttt{WithdrawTx}_i$, is a transaction that allows Carol to take an arbitrary amount of money out the channel. This amount is send to an arbitrary address specified by Carol. If Carol wants to withdraw the channel, she has to ask Bob for his cooperation. This transaction is not auto-generated when Carol send money to Bob, both have to be online to create this transaction.

This transaction automatically settle the channel with the amount owned by Bob at $\texttt{Channel}_{i,n}$ and changes the remaining amount available for Carol for the state $\texttt{Channel}_{i+1,n}$, thus, this remaining funds are moved to the next channel address.

Closed Channel Transaction The close channel transaction, hereinafter also mentioned as $\texttt{ClosedChannelTx}_i$, is also a cooperative transaction, that allows Carol or Bob to close the channel in the most effective way (less fee and quiker). This transaction has two output, one for Bob with the amount owned by Bob to his address and a second one for Carol with the remaining amount of money in the channel. When a $\texttt{ClosedChannelTx}_i$ is created, no more transaction must be created or accepted on the channel.

Pay To Channel Transaction The pay to channel transaction, hereinafter $\texttt{PayToChannelTx}_i$, allows Carol or Bob to send money directly to the current channel address. This transaction is usefull for Carol if there is not enough money onto the channel and she wants to send more money to Bob without opening an other payment channel. It is also usefull for Bob in the case he want to send money to Carol—he can send money directly to a Carol's address, but the payment can be related to a channel event or action—and allows her to reuse it into the channel, e.g fidelity points.

Before broadcasting the pay to channel transaction or before accepting that payment as part of the usable funds for Carol, an interim refund transaction needs to be created. This interim refund transaction is a safty garauntee for Carol until the merge occures.

For a state $\texttt{Channel}_{i,n}$ without any pay to channel transaction, the multisig address have, normaly, one unspend output. This unspend output is used as an input in each transactions and these transactions split it to track the balances of each parties. After a pay to channel transaction the multisig address have more than one unspend output. When Carol sends money to Bob they have to check if a pay to channel transaction occured and if it is the case they need to merge the interim refund and use all the unspend outputs. It is worth noting that the more pay to channel transactions occurred the more expensive the transaction become.

Interim Refund Transaction The interim refund transaction, hereinafter $\texttt{InterimRefundTx}_{i,n}$, is a temporary transaction used by Carol to get her money out of the channel. This transaction is created to protect Carol from Bob invalidating the current refund transaction.

Definition 3. *A channel merge occured each time a interim refund transaction is merged into the regular refund transaction and the regular settlement transaction.*

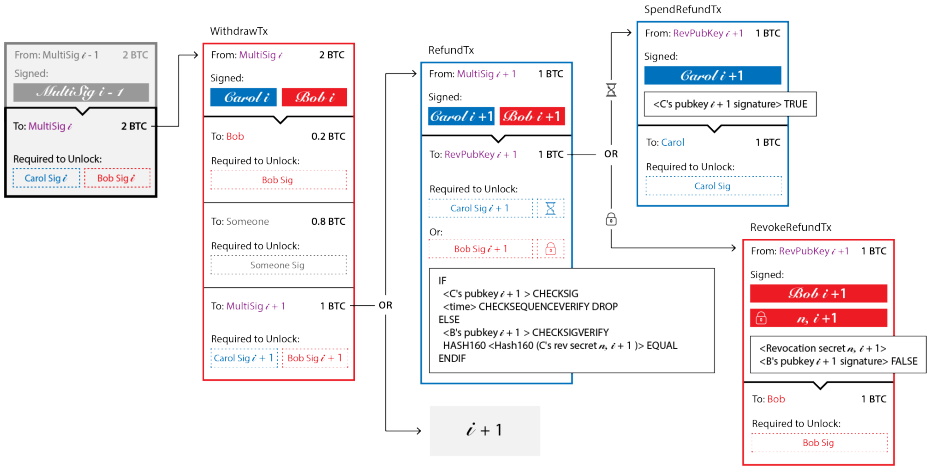

Fig. 4. Withdraw transaction that allows Carol to takes money out the channel, pay Bob, and move the remaining funds into the next multisig address. A refund transaction is created to allow Carol to recover her funds after the withdraw if no transaction is made. The refund transaction can be spent by Carol with a spend refund transaction and cannot be contested with the revoke refund transaction if no other transaction is made. If the state move to $\texttt{Channel}_{i+1,n+1}$, again, the secret for $\texttt{Channel}_{i,n-1}$ is shared, then Bob knows $\Phi_{i,n-1}(\texttt{Channel}_{i+1,n+1}) = \Phi_{i+1,n}$ and can revoke.

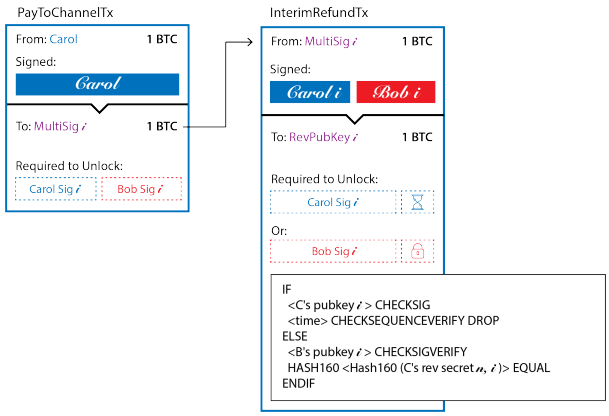

Fig. 5. Pay to channel transaction by Carol with the interim refund transaction. The interim refund transaction acts like a standard refund transaction but aims to be merged in the next round of transaction. The interim refund transaction has the same requirements to be spent than a standard refund transaction, if no transaction is made Carol can spend the interim refund, otherwise Bob can revoke the interim refund.

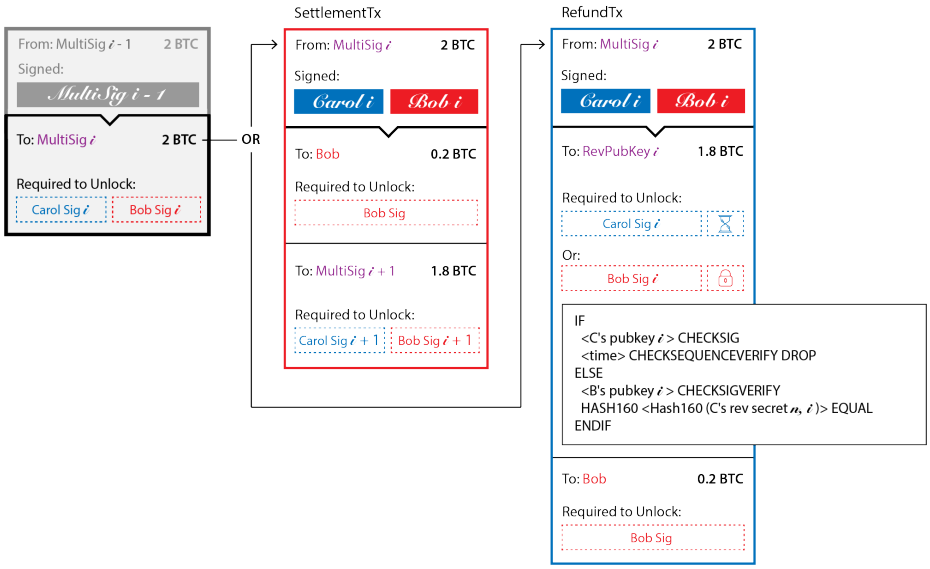

Fig. 6. Simplified view of possibilites for a standard state $\texttt{Channel}_{i,n}$ without second layer dependency transactions like spend and revoke. The content of a multisig can be settle by Bob (in red) or can be refund by Carol (in blue).

Definition 4. *A channel reduce occured each time a non-closing channel transaction is broadcast and included in the blockchain. The channel is then in reduced mode when one and only one* `UTXO` *is available in the current multisig address.*

3 Trustless One-way Payment Channel

3.1 Channel Setup

Before opening the channel Carol and Bob need to exchange keys for the channel account a and negotiate the relative timelock value.

1. Carol:
   (a) sends a request to open a channel with:
      i. the account a
      ii. the Carol's $\texttt{xPub}_a$
      iii. the relative timelock parameter
2. Bob:
   (a) if Bob agree with the request and timelock is whithin acceptable range, respond with:
      i. the Bob's $\texttt{xPub}_a$

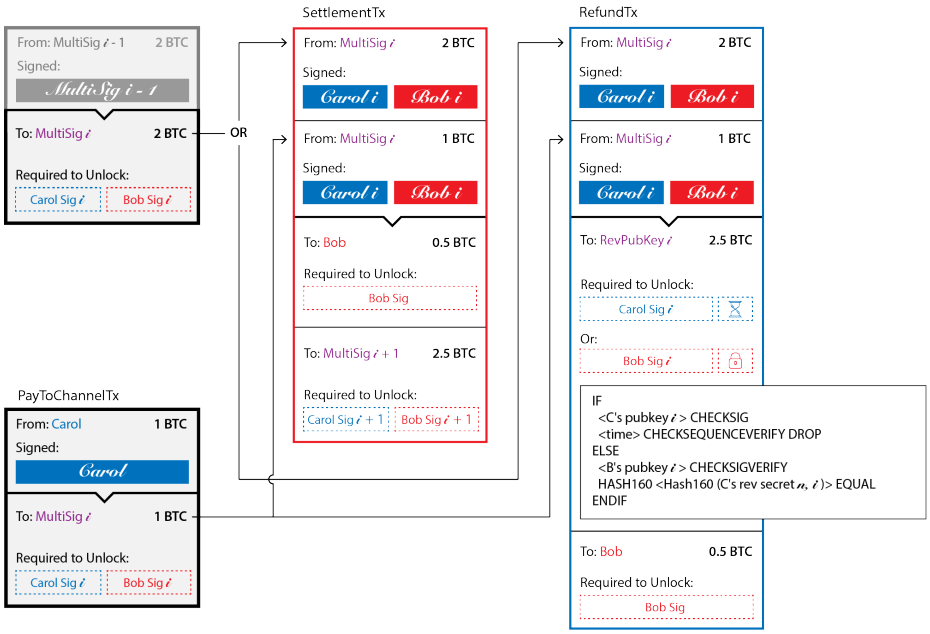

Fig. 7. Result of a merged pay to channel transaction after Carol sends 0.3 more Bitcoin to Bob. The $\texttt{Multisig}_i$ contains two `UTXOs`, (i) from the funding transaction or the last move from $\texttt{Multisig}_{i-1}$, and (ii) from the pay to channel transaction adding 1 BTC into the channel. Both settlement transaction and refund transaction contain the two `UTXOs` as input to sign and spend the totality with the adjusted balances.

3.2 Channel Opening

To open the channel, Carol and Bob have to cooperate to generate and fund a mutli-signatures address, hereinafter also mentioned as $\texttt{Multisig}_i$ address. This multi-signature address acts as the channel address and stores the totality of the channel's funds. This address holds normaly only one `UTXO`, but with pay to channel transactions, this could not be true.

1. Bob:
   (a) generates the $\texttt{Multisig}_i$ with Bob's Π_i and Carol's π_i and sends it
2. Carol:
   (a) create the $\texttt{FundingTx}^i_{<>}$ that funds the $\texttt{Multisig}_i$ address
   (b) generate $\Phi_{i,n}$
   (c) create $\texttt{RefundTx}^{i,n}_{<>}$ with $\texttt{Multisig}_i$ and $\Phi_{i,n}$ sending the full amount back to herself via the $\texttt{RevPubKey}_{i,n}$ contract
   (d) initiates the channel by sending:
      i. $\texttt{hash}(\Phi_{i,n})$

ii. $\texttt{RefundTx}^{i,n}_{<>}$

3. Bob:
   (a) receives the $\texttt{RefundTx}^{i,n}_{<>}$, signs it and returns $\texttt{RefundTx}^{i,n}_{<bob>}$
4. Carol:
   (a) broadcasts $\texttt{FundingTx}^{i}_{<carol>}$
5. Bob:
   (a) wait for transaction's confirmations
   (b) consider the channel as open

If Bob stops responding after the step 2, Carol has created transactions but has not use them. If Carol stops responding after step 3, Bob has signed a transaction who will maybe never been used. After a while, Bob must consider the channel opening as failed. If Bob stops responding after the setp 4, Carol can broadcast her refund transaction and she is safe. If Carol stops responding after opening the channel Bob do not lose anything.

3.3 Transact

Carol to Bob The Carol to Bob protocol allows Carol to send an arbitrary amount of money throught the channel. Carol desires to authorize a payment of M satoshis to Bob at $\texttt{Channel}_{i,n}$ state.

If there is a Bod to Carol transaction before and we need to use it because there is no more funds in the regular refund, we have to merge that refund in that new $\texttt{Channel}_{i,n+1}$ state. Bob can trust this unconfirmed output because it comes from himself.

1. Carol:
   (a) derives $\Phi_{i,n+1}$ and $\Phi_{i+1,n+1}$
   (b) generates the $\texttt{RefundTx}^{i,n+1}_{<>}$ with two outputs:
      i. Refund Output: Carol's new balance to $\texttt{RevPubKey}_{i,n+1}$ contract
      ii. Settlement Output: Bob's new balance to settlement address
   (c) sends a message to Bob containing:
      i. $\texttt{RefundTx}^{i,n+1}_{<>}$
      ii. $\texttt{hash}(\Phi_{i,n+1})$ and $\texttt{hash}(\Phi_{i+1,n+1})$
      iii. the amount of M satoshis being paid
2. Bob:
   (a) generates the $\texttt{SettlementTx}^{i}_{<>}$ with two outputs:
      i. Settlement Output: Bob's new balance
      ii. Change Output: Carol's new balance to $\texttt{Multisig}_{i+1}$ with Bob's Π_{i+1} and Carol's π_{i+1}
   (b) generates the $\texttt{PostSettlementRefundTx}^{i+1,n+1}_{<bob>}$ with:

i. Refund Output: sends Carol's funds to the associated $\texttt{RevPubKey}_{i+1,n+1}$ contract with the secret = $\texttt{hash}(\Phi_{i+1,n+1})$

(c) sends:

i. $\texttt{RefundTx}^{i,n+1}_{<bob>}$

ii. $\texttt{SettlementTx}^{i}_{<>}$

iii. $\texttt{PostSettlementRefundTx}^{i+1,n+1}_{<bob>}$

3. Carol:

(a) sends:

i. $\texttt{SettlementTx}^{i}_{<carol>}$

ii. the shared secret $\Phi_{i,n}$

4. Bob:

(a) upates state channel to $\texttt{Channel}_{i,n} \Rightarrow \texttt{Channel}_{i,n+1}$ and the payment can now be considered as final

If Bob stops responding after step 2, Carol can broadcast the refund transaction but she has no incentives to do that because she will go down her balance without counter-party. Because she has not yet share the secret $\Phi_{i,n}$, Bob cannot yet revoke the current $\texttt{Channel}_{i,n}$ state. If Carol does not respond at the step 3, Bob can settle the current $\texttt{Channel}_{i,n}$ state, but cannot settle the $\texttt{Channel}_{i,n+1}$ state in negotiation, Carol is safe. After step 3, Carol can refund herself and Bob can revoke the old $\texttt{Channel}_{i,n}$ state and settle the new $\texttt{Channel}_{i,n+1}$ state, the transaction is complete.

Channel Topop The channel topop protocol allows Bob or Carol to send an output directly to the current channel multisig address and allows Carol to include this output as part of usable funds immediatley. If the funds comes from Carol, they can be immediatley used for a withdraw transaction only.

To protect Carol, the refund for this additional amount is separate to the existing refund output to prevent Bob from invalidating Carol's refund transaction by sending an output which becomes invalid or not accepted by the network (lower fee, double spend, invalid script, etc.)

In subsequent transactions, once this output has confirmed, the refund should be merged into a single refund output as before, to be more efficient with refund transaction size.

1. Initiator:

(a) create the $\texttt{PayToChannelTx}^{i}_{<initiator>}$ that funds the $\texttt{Multisig}_i$ address

(b) create $\texttt{InterimRefundTx}^{i,n}_{<>}$ with $\texttt{Multisig}_i$ and $\texttt{hash}(\Phi_{i,n})$ sending the full amount back to Carol via the $\texttt{RevPubKey}_{i,n}$ contract

(c) sends:

i. $\texttt{InterimRefundTx}^{i,n}_{<initiator>}$

2. Receiver:
   (a) validate $\texttt{InterimRefundTx}^{i,n}_{<initiator>}$
   (b) sends if the payment is accepted or not
3. Initiator:
   (a) if the payment is accepted broadcast $\texttt{PayToChannelTx}^{i}_{<initiator>}$
4. Receiver:
   (a) wait for $\texttt{PayToChannelTx}^{i}_{<initiator>}$ transaction's confirmations

If the receiver does not validate the payment the initiator has no incentives to broadcast the transaction, if it is accepted, then the initiator can send money into the channel safely because of the interim refund transaction. Without negotiating a new state, Bob cannot revoke $\texttt{InterimRefundTx}^{i,n}_{<initiator>}$ and Carol can spend the refund. When a new $\texttt{Channel}_{i,n+1}$ state is negotiated, Bob can revoke the $\texttt{InterimRefundTx}^{i,n}_{<initiator>}$ if Carol try to broadcast it. At $\texttt{Channel}_{i,n+1}$, the refund transaction and the settlement transaction contain the merged refund transaction.

It is worth noting that the initiator does need to know the secret to create the pay to channel transaction and the interim refund transaction. An external player can ask the needed information to topop the channel knowing only public information.

Withdrawing The withdraw protocol allows Bob to authorize a withdrawal of M satoshis at Carol's request and with her cooperation for $\texttt{Channel}_{i,n}$ state. Bob needs to validate the withdrawal amount and can set up a set of rules internally to manage the channel economics. It is worth noting that when the withdraw takes action Bob automatically settle his funds without paying fee.

1. Carol:
   (a) derives $\Phi_{i+1,n}$
   (b) generates the $\texttt{Multisig}_{i+1}$ address with Bob's π_{i+1} and Carol's Π_{i+1}
   (c) generates the $\texttt{WithdrawTx}^{i}_{<>}$ with:
      i. Settlement Output: sends Bob's funds to the settlement address
      ii. Withdraw Output: withdrawal amount M to specified address
      iii. Change Output: new balance into $\texttt{Multisig}_{i+1}$
   (d) generates the $\texttt{RefundTx}^{i+1,n}_{<>}$ from address $\texttt{Multisig}_{i+1}$ with:
      i. Refund Output: Carol's new balance into $\texttt{RevPubKey}_{i+1,n}$ contract with the secret = $\texttt{hash}(\Phi_{i+1,n})$
   (e) sends:
      i. $\texttt{RefundTx}^{i+1,n}_{<>}$
      ii. $\texttt{WithdrawTx}^{i}_{<>}$
      iii. $\texttt{hash}(\Phi_{i+1,n})$

2. Bob:
   (a) verifies, signs and returns:
      i. $\texttt{RefundTx}^{i+1,n}_{<bob>}$
      ii. $\texttt{WithdrawTx}^{i}_{<bob>}$
3. Carol:
   (a) shares:
      i. $\Phi_{i,n}$ to invalidate the current state
      ii. $\texttt{WithdrawTx}^{i}_{<bob,carol>}$
4. Bob:
   (a) broadcast $\texttt{WithdrawTx}^{i}_{<bob,carol>}$
   (b) upates state channel to $\texttt{Channel}_{i,n} \Rightarrow \texttt{Channel}_{i+1,n}$ and validate exchange

If Bob does not respond in step 2, Carol has not disclosed any important informations. If Bob stops responding after step 2, Carol can withdraw the amount and safely refund her funds if no transaction is negotiated. If Carol does not respond after the step 2, Bob must wait a while and if the withdraw transaction is not broadcasted, he must broadcast the settlement transaction to force the transition to the next $\texttt{Channel}_{i+1,n}$ state.

Settling The settling protocol allows Bob to broadcast at $\texttt{Channel}_{i,n}$ state the $\texttt{SettlementTx}_{i,n}$ to get the settlement output and move the remaining funds into the next $\texttt{Multisig}_{i+1}$ address. In this case the channel stays open and Carol can create new transactions or close the channel.

If the $\texttt{SettlementTx}_{i,n}$ is broadcasted and Carol wants to close the channel, she can broadcast the $\texttt{PostSettlementRefundTx}_{i,n}$ and wait the timelock to get her money back. Carol has to query the network to know if the $\texttt{SettlementTx}_{i,n}$ is broadcast, she can only query the blockchain before each new transaction to be sure that the settlement transaction has not be broadcasted yet.

3.4 Channel Closing

Cooperative Closing cooperatively the channel allows Carol—or Bob if Carol is online—to ask if Bob agrees to close efficiently the channel, withdrawing the full remaining balance, at $\texttt{Channel}_{i,n}$ state. The following steps 3 and 4 can be merge and executed by the same player depending the implementation.

1. Carol:
   (a) generates the $\texttt{ClosedChannelTx}^{n+1}_{<carol>}$ with:
      i. Settlement Output: sends Bob's funds to Bob address
      ii. Change Output: sends Carol's funds to Carol address
   (b) sends $\texttt{ClosedChannelTx}^{n+1}_{<carol>}$

2. Bob:
   (a) verifies and signs $\texttt{ClosedChannelTx}^{n+1}_{<carol>}$
   (b) sends $\texttt{ClosedChannelTx}^{n+1}_{<carol,bob>}$
3. Carol:
   (a) broadcasts $\texttt{ClosedChannelTx}^{n+1}_{<carol,bob>}$

Contentious The contentious channel closing protocol allows Carol to close the channel alone, i.e., without Bob's cooperation or response, at $\texttt{Channel}_{i,n}$ state. Carol can broadcast her fully signed refund transaction sending her owned funds to $\texttt{RevPubKey}_{i,n}$ address. Carol would then need to spend from the revocation public key contract after the timelock delay with the spend refund transaction.

It is worth noting that only Carol can close the channel, but Bob can get his money by broacasting his settlement transaction at any time.

4 Evidence of Trustlessness

In the following, axioms, possible edge-cases, and discovered attacks, with an evidence of trustlessness for the channel protocol, are exposed. *Liveness* in the blockchain, i.e. transactions can be included in the next blocks, is asumed to guarantee the security model.

4.1 Axioms

Refund Transaction For $\texttt{Channel}_{i,n}$ state, if Carol broadcasts the current refund transaction, Bob cannot revoke it without knowing $\Phi_{i,n}$. After the timelock, Carol can generate and broadcast a spend refund transaction.

$$\forall i \wedge n, \exists \Phi_{i,n}: \quad \forall \Phi_{i,n}, \text{bob knows } \Phi_{i,n-1} \wedge \texttt{hash}(\Phi_{i,n})$$

The same rule is applied to interim refund transactions, if Carol broadcast the current interim refund transaction, Bob cannot revoke it without knowing $\Phi_{i,n}$, and Carol can spend the interim refund after the timelock.

Old Refund Transaction For $\texttt{Channel}_{i,n}$ state, if Carol broadcasts an old refund transaction, e.g. $n-1$, then Bob has a the time during the timelock to generate and broadcast the revoke transaction for the state $\texttt{Channel}_{i,n-1}$ with $\Phi_{i,n-1}$ secret.

$$\forall 0 \leq x < n, \exists \Phi_{i,x}: \quad \forall \Phi_{i,x}, \exists \texttt{RevokeTx}^{i,x}_{<bob>}$$

The same rule is applied to old interim refund transactions, if Carol broadcast an old interim refund transaction, e.g. $n-1$, Bob can revoke it with $\Phi_{i,n-1}$ secret.

Settlement Transaction For $\texttt{Channel}_{i,n}$ state, if Bob broadcasts his knowed $\texttt{SettlementTx}_{i,n}$ transaction, Carol has the choice to close the channel or transact on top of the new $\texttt{Multisig}_{i+1}$ address.

$$\forall \mathcal{C} = \texttt{Channel}_{i,n}, \exists \Phi_{i,n} \wedge \Phi_{i+1,n} : \quad \text{bob knows } \Phi_{i+1,n} \text{ iff } \mathcal{C} = \texttt{Channel}_{i+1,n+1}$$

Contentious Channel Closing By contentious it means that all players are not communicating anymore and/or do not agree on a valid state. Let's define the way for Carol to close the $\texttt{Channel}_{i,n}$ state.

$$\forall \texttt{Channel}_{i,n}, \exists \texttt{RefundTx}^{i,n}_{<carol,bob>} \text{only owned by Carol}$$

and

$$\forall \texttt{RefundTx}^{i,n}_{<carol,bob>}, \exists \texttt{SpendRefundTx}^{i,n}_{<carol>} \text{only owned by Carol}$$

so:

$$\forall \texttt{Channel}_{i,n}, \exists \texttt{SpendRefundTx}^{i,n}_{<carol>} \text{only owned by Carol}$$

4.2 Edge Cases

Someone does not broadcast Cooperative Transaction If one player does not share a fully signed cooperative transaction and the secret $\Phi_{i,n}$ attached to the current $\texttt{Channel}_{i,n}$ state, then the other player need to force after a while the transition into the new $\texttt{Channel}_{i+1,n}$ state with his own fully signed transaction, i.e $\texttt{RefundTx}_{i,n}$ or $\texttt{SettlementTx}_{i,n}$ transaction.

4.3 Attacks

In this section, attacks' vector discovered and fixes are discussed. Attacks exposed are no longer valid in the current scheme, but a deep analysis has been carry out to generalize the protocol construction and improve the scheme.

Old Settlement Attack With Weak Secret It is possible for Bob to lock the funds into the multisig or steal the money if the secret construction is too weak. For a channel at N dimensions the secret is considered weak if:

$$|\Phi| < N$$

Let's assume that the revocation secret Φ depends only on n and not on i for $\texttt{Channel}_{i,n}$. Thus, the secret can be expressed by:

$$|\texttt{Channel}_{i,n}| = N = 2 : |\Phi_n| = 1 \implies |\Phi_n| < N$$

Then, for $\texttt{Channel}_{i,n}$, if Bob broadcast an old settlement transaction, e.g. $n-1$, then Carol cannot use her post settlement refund transaction because

she previously shared the secret Φ_{n-1}. So the remaining funds are blocked in the $\texttt{Multisig}_{i+1}$ address. To be able to get her funds back, Carol would have to transact with Bob, if Bob does not cooperate, Carol has no way to recover her funds. If she try to refund the $\texttt{Multisig}_{i+1}$, then Bob can revoke with Φ_{n-1} secret.

If the secret dimension is equal to the channel dimension, i.e. $|\Phi_{i,n}| = |\texttt{Channel}_{i,n}|$, then the previous shared secret is $\Phi_{i,n-1}$ and the secret for refunding the $\texttt{Multisig}_{i+1}$ address at $\texttt{Channel}_{i,n-1}$ state is $\Phi_{i+1,n-1}$ and then:

$$\Phi_{i,n-1} \neq \Phi_{i+1,n-1}$$

Game theory is not sufficient to ensure the security of the channel if, when a player acts dishonestly, it exists an incentive to gain, even probabilistically, over the other player. In this case, the provider lose funds by broadcasting the $\texttt{Channel}_{i,n-1}$ state but can gain all funds if the client does not act correctly and does unlock his funds.

5 Further Improvements

Improvements can be done in two ways: (i) extending channel capabilities or (ii) optimizing the channel costs by reducing the transaction size or their number.

5.1 Threshold Signatures

The abilities of settle and withdraw the channel without closing has a downside, each time a transaction is broadcasted on chain and cost fees. Optimizing the channel transaction size or the number of transaction needed is an area of further research.

The principal cost of a transaction come from it inputs and their type. A channel transaction spend one or more `UTXOs` from the $\texttt{Multisig}_i$ address. These `UTXOs` are `P2SH` of a Bitcoin 2-out-of-2 multi-signature script that requires, obviously, two signatures. Knowing that a signature size is at least 64 bytes and an average transaction size (one simple input and one or two outputs) is a bit more than 200 bytes, it is easy to see that replacing the `P2SH` with a 2-out-of-2 multisig `UTXOs` by `P2PKH` `UTXOs` is more efficient in any cases.

To achive it, an ECDSA threshold signature scheme, with the same requirements as the 2-out-of-2 multisig, is required. This scheme exists and can be adapted from the paper "Two-Party Generation of DSA Signatures" by MacKenzie and Reiter [4].

5.2 Pre-authorized Payments

Pre-authorized payments are required in other real case scenario like provider acting as a payment processor. The client must be able to set a limit in which the provider can take the money during a time limit.

Further research can be done in this area to figure out the achievability and the most effective way to implement this feature into this scheme. May be a third layer, on top of layer's two, is necessary and achievable, may be the channel dimension can be increased.

6 Related Work

Simple micropayment channels were introduced by Hearn and Spilman [3]. The Lightning Network by Poon and Dryja [6], also creates a duplex micropayment channel. However it requires exchanging keying material for each update in the channels, which results in either massive storage or computational requirements in order to invalidate previous transactions. Finally, Decker and Wattenhofer introduced a payment network with duplex micropayment channels [2].

7 Conclusion

Trustless one-way payment channel for Bitcoin resolves many problems. Scalability is near infinite and costs of the channel decrease linearly with the number of transaction present into the channel. Delays to consider a transaction as valid are brought back to network delay and minimal check time. Clients do not need to be online to keep their funds safe and can withdraw arbitrary amount and refill the channel at any time. The provider does not need to lock funds to receive money and the cost of a channel setup is low.

8 Acknowledgement

Loan Ventura and Thomas Roulin are acknowledged for their helpful contribution and comments during the completion of this work.

References

[1] Ryan X. Charles and Clemens Ley. *Yours Lightning Protocol*. 2016. URL: `https://github.com/yoursnetwork/yours-channels/blob/master/docs/yours-lightning.md`.

[2] Christian Decker and Roger Wattenhofer. "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels". In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Andrzej Pelc and Alexander A. Schwarzmann. Cham: Springer International Publishing, 2015, pp. 3–18. ISBN: 978-3-319-21741-3.

[3] Mike Hearn and Jeremy Spilman. *Bitcoin contracts*. URL: `https://en.bitcoin.it/wiki/Contracts`.

[4] Philip MacKenzie and Michael K. Reiter. "Two-Party Generation of DSA Signatures". In: *Advances in Cryptology — CRYPTO 2001*. Ed. by Joe Kilian. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 137–154. ISBN: 978-3-540-44647-7.

[5] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: `http://bitcoin.org/bitcoin.pdf`.
[6] Joseph Poon and Thaddeus Dryja. *The bitcoin lightning network*.

A Transaction Dependency Graph