Partially Non-Interactive and Instantaneous Bitcoin One-way Payment Channel

October 17, 2018 – DRAFT – THIS DOCUMENT IS CONFIDENTIAL AND UNDER NDA – NO COPY ALLOWED

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

TrueLevel SA, Neuchâtel, Switzerland
{tom, joel, d}@truelevel.io

Abstract. The most significant challenge for Bitcoin in the coming years is scalability. Currently, Bitcoin enforces a 1 Megabyte block-size limit. On average every 10 minutes a new block is discovered. That produces a payment network limited to ≈ 7 transactions per second, and with delayed (and unknown) transaction confirmation times. This is not sufficient in comparison to the currently deployed payment infrastructure such as credit card processors, which allows tens of thousands of transactions per second. To address this, there are some proposals to modify (i) the transaction structure (such as in SegWit), (ii) the block-size limit (such as SegWit2x) or even (iii) build a second layer on top of the Bitcoin protocol (such as the Lightning Network). Following the rational of second layer solutions, we propose a retail-ready, one-way payment channel that enables two parties to transact mostly off-chain, thus reducing the number of on-chain transactions and operational cost, while remaining secure and trustless. After the channel is opened for a few blocks, payment channel transactions are instantaneous and irreversible, and do not rely on the seemly random block arrival times. At all times, only one of the parties (the money receiver) must stay online to secure the channel integrity. The money receiver can redeem his channel funds on chain with no delay at all times. The money sender does not have to perform any action to secure its funds.

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

1 Introduction

Decentralized crypto-currencies such as Bitcoin [8] and its derivatives employ a special decentralized public append-only log based on proof-of-work called the *blockchain*. In a decentralized crypto-currency, users transfer their funds by publishing digitally signed transactions. Transactions are confirmed only when they are included in a block that extends the longest chain, which is validated and accepted by other nodes of the network. To get the right to include a block to the blockchain, bitcoin miners solve a proof-of-work problem that can only

be solved by brute force computation. For its work, the miner has the right to create some bitcoins. And this is the only source of bitcoins in the network, thus all bitcoins can be tracked down to its creation by miners.

The blockchain protects against state transitions that are conflicting with each other, for example *double-spending* attempts. Since the money is digital, nothing prevents a user to send the same digital funds to two different recipients at the same time. Although a malicious bitcoin owner potentially 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. The whole history all coins, from creation by miners to the present state must be unambiguous, and verifiable by all nodes.

The bitcoin blockchain is slow, growing on average at 1 MB per 10 min velocity. Most of its security model depends on transactions getting included to the blockchain within a certain amount of time. Thus a block space market developed to ensure that transactions get confirmed on chain. Users pay a variable miners fee to get their transactions prioritized by the miners. In December 2017, a large fraction of users payed over USD $30 in fees to get their transactions quickly confirmed.

Scalability is one of the most significant challenges in blockchain systems. As mentioned before, some proposals are focused on the blockchain data [10, 5, 11] structures and the consensus layer, others [9] are focused on a second layer of transactions where transactions are created off-chain and the blockchain itself is used as a conflict resolution system and source of truth. These proposals are called payment channels, or layer-two applications, and provide a wide range of advantages. The idea of payment channels was suggested by Satoshi in an email to Mike Hearn. Since then, various propositions to construct such structures have been proposed.

1.1 The value of unidirectional channels

Streaming payments in a retail context are mostly unidirectional. As an example, most people receive their salaries once a month—a large incoming payment—and across the rest of the month they send small outgoing payments to cover their living costs.

Thus on average the number of incoming transactions, $N(tx_{in})$, is far lesser than the number of outgoing transactions, $N(tx_{out})$

$$N(tx_{in}) \ll N(tx_{out})$$

In bitcoin, the transaction cost is related to the size of the transaction in bytes and not the amount transacted in bitcoin. Thus small or large payments incurs approximately the same transaction cost.

When decreasing the number of on-chain payments, the most weighted variable thus is $N(tx_{out})$, and it should be minimized. Interestingly, unidirectional payment channels do minimize $N(tx_{out})$, although they have no effect on $N(tx_{in})$, in the case.

However, the previous statement assumes that all payments go to the same recipient. Thus here the client, hereinafter Carol (the money sender), wants to buy goods or services from a business, Bob (the money receiver). For a channel to generate savings, Carol and Bob should have a lasting relationship. For example, Bob sells goods or services to Carol repeatedly within a small time interval.

1.2 The issues of bidirectional channels

Bidirectional payment channels impose the burden upon both parties to police the channel by listening to the network at all times and submitting transactions to enforce the correct state gets on-chain [9], which greatly increases the complexity for the both participants. This property make such channels less practical for real world retail use cases.

1.3 Wish-list for a channel to be used in retail settings

On this work, we will focus on a partially non-interactive (for the client), partially instantaneous (for the business), one-way channel that is more suitable for the retail context (loosely based on [2, 9]).

It is of general believe that bitcoin cannot be used for retail. Transaction cost is too high and confirmation times are too long. As it is popularly stated "You cannot pay for your coffee with bitcoin!" And in 10 years time, why should anyone have to validate your coffee payment?

With that in mind, we acknowledge the following constraints to be essential for a channel to become practical for usage in retail settings.

(i) Channel transactions finality must be reached instantaneously—It must not require waiting for a block to be mined. That is, Bob knows for sure that he received a payment, and that it is irreversible, and should feel confident to handle the products to Carol and let her walk off his store.

(ii) Few on-chain transactions—Cheap to make large numbers of small payments with only a couple of on-chain transactions. Additionally, on-channel transactions are private. Thus the 10-year old coffee purchase will be hidden together with other purchases on a single bitcoin transaction.

(iii) No counter-party risk—Any party can disappear from existence and the other party's funds stay safe, and are redeemable at any time. Each party must be able to single-handedly get their unspent or received funds by publishing on chain transactions at all times without the need of the other party to cooperate.

(iv) Bob must be able to settle at any time, and immediately be able to spend the funds—the funds owed to Bob are fully liquid at all times—The goods and services were provided to Carol, but Bob's money is locked up in the channel and he should be able to redeem his funds with no delay whenever he needs it to provide liquidity to his business.

(v) Bob must not lock funds upfront for each of the clients—It is unfeasible and costly for retail businesses to have to stake funds for each of their several clients.

(vi) Carol does not have to watch over her deposited funds—She deposits the money and forgets about it. It is safe in there at all times.

The currently available proposals of bidirectional channel do not fulfill the last 3 constraints of the list, and therefore are unsuitable for the task. On this work, we specifically designed a payment channel construct that fulfills the entire list of the above constraints. To achieve that we focused on a unidirectional payment channel. Further characteristics of the channel are listed below.

(i) The channel should stay open for an unlimited amount of time, like a checking bank account, and Carol has the option to close the account at any time and take its funds back.

(ii) Carol, who has locked funds in a channel with Bob, must be able to withdraw an arbitrary amount out of her channel balance without closing it, with the Bob's cooperation. In case Bob does not cooperate, then she has to close the channel.

(iii) Bob is expected to stay online to stay safe. He runs a business afterall and can be expected to incur the cost to secure his funds.

(iv) If Bob needs to send money to some clients, it is assumed that these transactions are regular on-chain transactions or via other channels.

1.4 Incentive structure

From a game theoretical perspective, Carol (or Bob) should always submit the transaction that pays herself (or himself) the most. On a one-way channel whereby Carol locked funds initially, the transaction that pays Carol the most is the first refund transaction—all the money goes back to her. While for Bob, the one that pays him the most will always be the last one—interestingly, by design the last state is always the valid one. This asymmetry promotes Bob to behave correctly and submit the last state even in the absence of policing by Carol. Thus the only party of the channel that must be policed is Carol, as she will always have the incentive to publish old state that pays her more. This feature is used to produce a practical channel in which clients do not have to stay online.

1.5 Transaction malleability

The scheme we present requires a proper fix to transaction malleability such as SegWit [10, 5]. It is assumed that a chain of unconfirmed transactions can be creates and trusted without breaking the security model defined above.

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, one at a time, with increments of +1 on i only or on n only. Each increment represents a state transition (or a move) in the channel state. Index i represents the offset of the multisig address where the channel's funds are locked. Index n represents the offset used to create the revocation secrets, which are later used in smart contracts to revoke the past off-chain transactions.

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 its life. We need to share public hierarchical deterministic addresses between the client and the server. Let's define the hierarchical 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 Π, 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:

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

Unlike Θ_n, the nodes Π_i, π_i, Ω_i and ω_i are not hardened derivations. It cannot be done because the public keys π_i and ω_i must be computed from $\texttt{xPub}_a$.

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, noted $\Phi_{i,n}$, corresponds to the state $\texttt{Channel}_{i,n}$ and depends on the secret Θ_n and the revocation key Ω_i.

$$\Phi_{i,n} = \texttt{HMAC}(\Theta_n, \Omega_i)$$

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 contracts are used in the payment channel scheme. The first 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 to 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 $\texttt{Channel}_{i,n}$, uses 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 case 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 $\texttt{Channel}_{i,n}$ state.

2.3 Transactions

A transaction is represented as $\texttt{Transaction}^{i,n}_{<>}$, where Transaction denotes the name of the transaction, the superscript indexes represent the transaction's state dependencies—on this example i and n—and the subscript represents who has already signed the transaction—denoted by the <> for no signatures. If a transaction is signed by Carol the transaction is noted $\texttt{Transaction}^{i,n}_{<carol>}$. Transactions that appear in blue on the figures are fully signed and only available to Carol and in red are fully signed and only available to Bob, thus only the specific parties can broadcast the transaction.

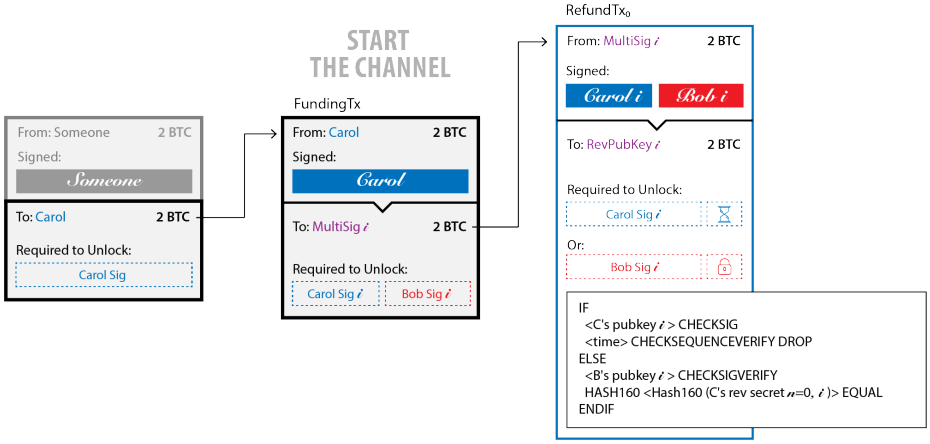

Fig. 1. Funding transaction that starts the channel by sending money in the first multisig address. Along with the first refund transaction that allows Carol to close the channel if no transactions are made.

Funding Transaction The funding transaction, hereinafter $\texttt{FundingTx}^{i}_{<>}$, is the transaction sending 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 until she possesses 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 channel transactions have ever been made, Bob cannot revoke the contract.

Refund Transaction The refund transaction, hereinafter $\texttt{RefundTx}^{i,n}_{<>}$, is a transaction that keeps track of the balances of Carol and Bob at $\texttt{Channel}_{i,n}$. It also allows Carol to close the channel if Bob does not respond or does not cooperate anymore. This transaction has one or more inputs and two outputs. The number of inputs depends on how many unspent transaction outputs are available at the $\texttt{Multisig}_i$ address. The first output represents the amount still owned by Carol, and the second, if present, is the amount owned by Bob. Bob might have a channel balance equal to zero, in which case the second output is excluded, such as when starting the channel, or right after Bob settles the channel.

Carol's balance is sent to a revocation contract corresponding to the channel state. This prevents Carol from broadcasting 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 broadcast by Carol so the fees are substracted from the first output, owned by Carol.

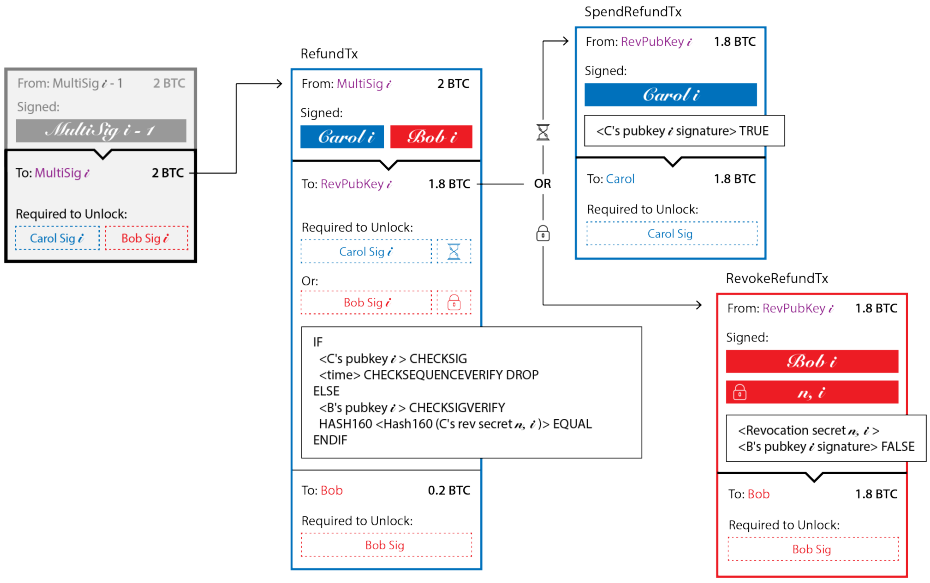

Fig. 2. Refund transaction based on the current multisig address with the associated spend and revoke transactions. The former allows Carol to get her money back and the latter Bob to revoke the contract if he knows the secret.

Because a refund transaction spends funds from a multisig address, it must be signed by both Carol and Bob to be considered valid. The revocation contract used in the Carol's output can be spent with a *spend-refund* transaction after a *timelock* delay. She only 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 signs with his Ω_i key.

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

A settlement transaction always has one output that sends Bob's balance directly to his address and one output that sends the remaining funds to the next multisig address $\texttt{Channel}_{i+1,n}$. Because the funds are sent 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, in this case, are already settled.

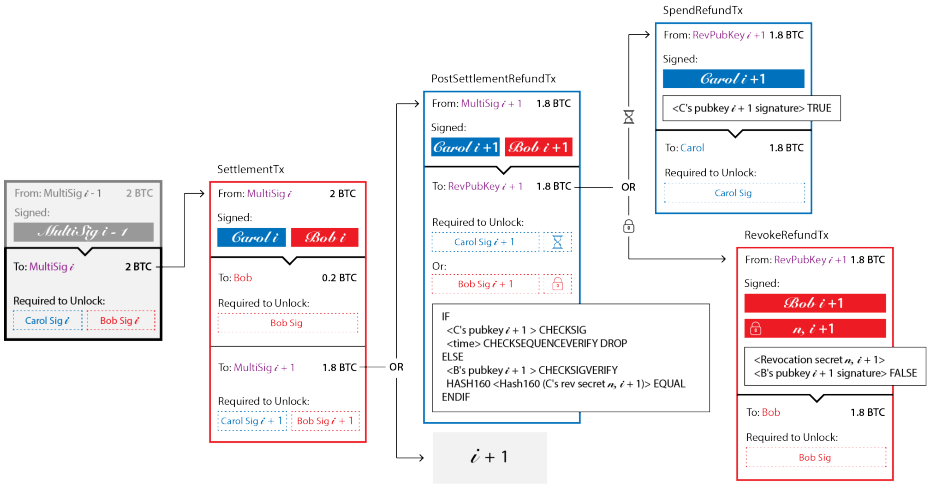

Fig. 3. Settlement transaction that allows Bob to settle the channel, moving the remaining funds to the next multisig address with the post settlement refund transaction. The latter transaction allows Carol to close the channel directly after the settlement.

If Bob broadcasts the fully signed settlement transaction, Carol has two options:

(i) continue to transact on the channel with the new multisig address; or

(ii) close the channel with her post settlement refund transaction.

It is worth noting that the secret for the revocation contract is $\Phi_{i+1,n}$, so in the case of a new channel 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 refund transaction aims 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 transactions are made after the settlement transaction. However, when Carol sends an amount to Bob, she shares the secret needed to revoke the contract. Therefore she cannot broadcast this transaction that is now attached to an old state.

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

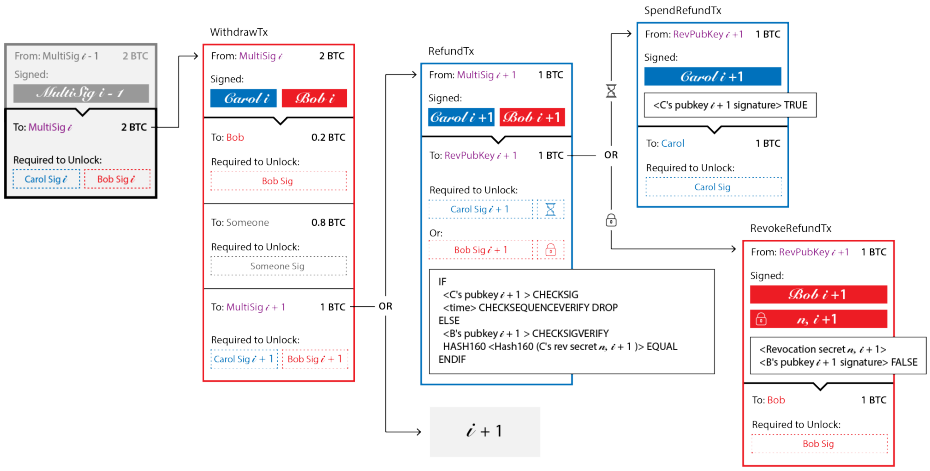

Fig. 4. Withdraw transaction that allows Carol to take money out the channel, pay Bob, and move the remaining funds into the next multisig address. A refund transaction is created to allow Carol to recover her funds after the withdraw if no transaction is made. The refund transaction can be spent by Carol with a spend refund transaction and cannot be contested with the revoke refund transaction if no other transaction is made. If the state moves 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.

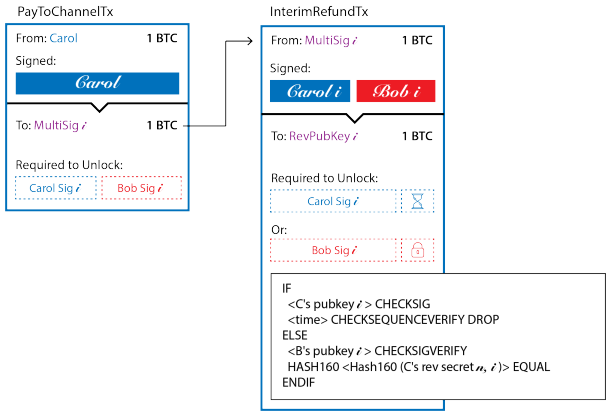

Fig. 5. Pay to channel transaction by Carol with the interim refund transaction. The interim refund transaction acts like a standard refund transaction but aims to be merged in the next round of transactions. The interim refund transaction has the same spending requirements as a standard refund transaction,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

This transaction automatically settles 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, 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 quicker). This transaction has two outputs, 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 transactions can 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 useful for Carol if there is not enough money on the channel and she wants to send more money to Bob without opening another payment channel. It is also useful for Bob in the case he wants to send her money and to allow her to reuse it in the channel. He could send money directly to Carol's address, but the payment might be related to a channel event or action, e.g fidelity points.

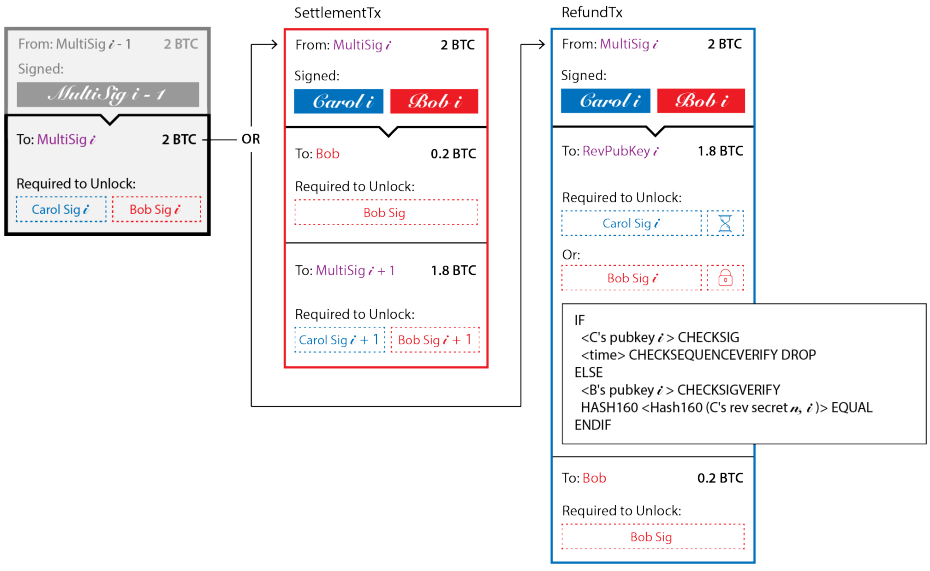

Fig. 6. Simplified view of possibilities for a standard state $\texttt{Channel}_{i,n}$ without second layer dependency transactions like spend and revoke. The content of a multisig can be settled by Bob or can be refunded to Carol.

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 safety guarantee for Carol until the merge occurs.

For a state $\texttt{Channel}_{i,n}$ without any pay to channel transaction, the multisig address usually has one unspent output. This unspent output is used as an input for each transaction and these transactions split it to track the balances of each party. After a pay to channel transaction, the multisig address has more than one unspent output. When Carol sends money to Bob they have to check if a pay to channel transaction occured. In this case they need to merge the interim refund and use all the unspent outputs. It is worth noting that the more pay to channel transactions occur the more (fee) cost incurred.

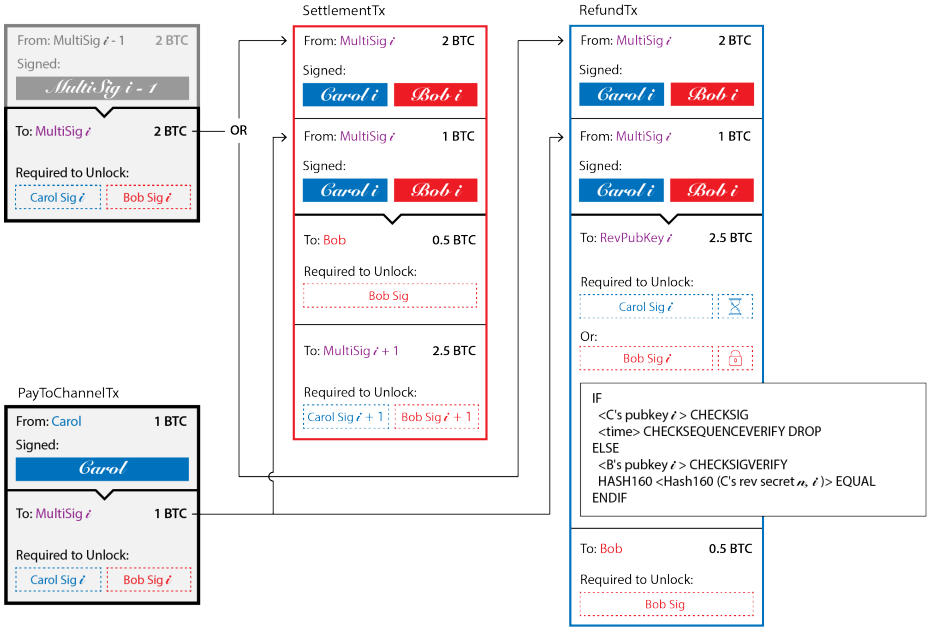

Fig. 7. Result of a merged pay to channel transaction after Carol sends more Bitcoin to Bob. The $\texttt{Multisig}_i$ contains two `UTXOs`, (i) from the funding transaction or the last move from $\texttt{Multisig}_{i-1}$, and (ii) from the pay to channel transaction adding 1 BTC into the channel. Both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

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.

If an unconfirmed Bob to Carol transaction exists and is needed because the regular multisig address is empty, then the refund must be merged into a new $\texttt{Channel}_{i,n+1}$ state. Bob can trust this unconfirmed output because it comes from himself.

Definition 1 (Channel merge). *A channel merge occurs each time an interim refund transaction is merged into the regular refund transaction and the regular settlement transaction.*

Definition 2 (Channel reduce). *A channel reduce occurs 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 Partially Non-Interactive and Instantaneous Channel

In the following, we describe the protocol layer. The protocol layer enforces the trustlessness of the channel for every player at each step. A player is safe as long as he follows this protocol.

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. Carol's $\texttt{xPub}_a$
      iii. the relative timelock parameter
2. Bob:
   (a) if Bob agrees with the request and timelock is whithin acceptable range, respond with:
      i. Bob's $\texttt{xPub}_a$

3.2 Channel Opening

To open the channel, Carol and Bob must cooperate to generate and fund a multi-signature address, hereinafter also referred to 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 generally holds only one `UTXO`, but with pay to channel transactions, this could be different.

1. Bob:
   (a) generates $\texttt{Multisig}_i$ with Bob's Π_i and Carol's π_i and sends it

2. Carol:
   (a) creates $\texttt{FundingTx}^{i}_{<>}$ that funds the $\texttt{Multisig}_i$ address
   (b) generates $\Phi_{i,n}$
   (c) creates $\texttt{RefundTx}^{i,n}_{<>}$ 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 $\texttt{RefundTx}^{i,n}_{<>}$, signs it and returns $\texttt{RefundTx}^{i,n}_{<bob>}$
4. Carol:
   (a) broadcasts $\texttt{FundingTx}^{i}_{<carol>}$
5. Bob:
   (a) waits for transaction's confirmations
   (b) consider the channel as open

If Bob stops responding after step 2, Carol has created transactions which she is unable to use. If Carol stops responding after step 3, Bob has signed a transaction which will probably never be used. After a while, Bob must consider the channel opening as failed. If Bob stops responding after step 4, Carol can broadcast her refund transaction and she is safe. If Carol stops responding after opening the channel Bob does not lose anything.

3.3 Transact

Carol to Bob The Carol to Bob protocol allows Carol to send an arbitrary amount of money through the channel, as far as the amount is smaller than or equal to the funds locked on the $\texttt{Multisig}_i$ address. Carol desires to authorize a payment of M satoshis to Bob at $\texttt{Channel}_{i,n}$ state.

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 lose part of her balance compared to broadcasting the previous state refund transaction.

Because she has not yet shared the secret $\Phi_{i,n}$, Bob cannot yet revoke the current $\texttt{Channel}_{i,n}$ state. If Carol does not respond at 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 Top-up The channel top-up 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 immediately if it is from Bob. If the funds come from Carol, they can be immediatley used for a withdraw transaction only.

To protect Carol, the refund for this additional amount is separated from 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 been 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 safely send money into the channel 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 tries 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 top-up 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 place Bob funds get automatically settled without him paying fees.

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 during step 2, Carol has not disclosed any important information. 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 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.

Settlement The settlement 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 broadcast 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 $\texttt{SettlementTx}_{i,n}$ has been broadcasted, she can only query the blockchain before each new transaction to be sure that the settlement transaction has not been broadcasted yet.

3.4 Channel Closing

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

For state $\texttt{Channel}_{i,n}$, Bob can broadcast his fully signed settlement transaction to get his money back, but Bob cannot close the channel. In that case Carol, if she want, can broadcast her fully signed post settlement transaction at any time to close the channel. Carol would then need to spend from $\texttt{RevPubKey}_{i,n}$ after the timelock delay.

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 is a prerequisite to guarantee the security model. That is, broadcasted transactions are assumed to get included in the blockchain within a predictable time delay.

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.

$$\forall(i,n)\exists\Phi_{i,n}|\text{Bob knows } \Phi_{i,n-1} \text{ and } \texttt{hash}(\Phi_{i,n})$$

The same rule is applied to interim refund transactions, if Carol broadcasts 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 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.

$$\forall x \in \{1, \ldots, n-1\} \exists \texttt{RevokeTx}^{i,x}_{<bob>}$$

The same rule is applied to old interim refund transactions, if Carol broadcasts 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 transaction $\texttt{SettlementTx}_{i,n}$, Carol has the choice to close the channel or transact on top of the new $\texttt{Multisig}_{i+1}$ address.

$$\forall \texttt{Channel}_{i,n} \exists \Phi_{i,n} \text{ and } \exists \Phi_{i+1,n}$$

$$\text{Bob knows } \Phi_{i+1,n} \iff \exists \texttt{Channel}_{i+1,n+1}$$

Contentious Channel Closing By contentious it is meant 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 eventually needs to force 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, discovered attack vectors and fixes are discussed. Attacks exposed are no longer valid in the current scheme, but a deep analysis has been carried 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 in the multisig or steal the money if the secret construction is too weak. For a N dimensions channel the secret is considered weak if

$$|\Phi| < N$$

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

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

Then, for $\texttt{Channel}_{i,n}$, if Bob broadcasts 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 tries to refund the $\texttt{Multisig}_{i+1}$, then Bob can revoke it with Φ_{n-1} secret.

If dimension of used secret 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 therefore

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

Game theory is not sufficient to ensure the security of the channel; if, when a player acts dishonestly, there exists an incentive to gain, even probabilistically, over the other player. In this case, the provider loses 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.

Secret Size Attack A recent advisory notes a vulnerability in the common construction of cross-chain smart contracts (contracts between distinct cryptocurrencies) through hash locking [6]. It focus on the primary use case in atomic swap contracts, which allow two cryptocurrencies to changes hands simultaneously and safely. The vulnerability can be generalized to a secret s and its hash $h = \texttt{hash}(s)$ generated by a player p_1 but required by another player p_2. When player p_1 send the hash h to player p_2, p_2 do not has the possibility to assert the validity of $h = \texttt{hash}(h)$.

Our scheme is not affected by this attack. In the protocol Carol generates the revocation secret, send the corresponding hash to Bob, and Bob assumes that a valid secret exists and is known by Carol. Bob use the received hash to create and validate the first transaction and accept the payment. When the second transaction occurs Bob receive the new hash and the previous secret. At this moment he can validate if a revocation secret exists and is valid, otherwise Bob refuses the new payment and Carol has no choice but closing the channel cooperatively or not.

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 occurences.

5.1 Threshold Signatures

The ability to settle and withdraw the channel without closing has a downside. Each time a transaction is broadcast on chain, a fee is charged. Optimizing the channel transaction size or the number of transactions needed is an area of further research.

The main cost of a transaction comes from its inputs and their types. A channel transaction spends 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 case.

Three transactions are compared with SegWit[1] and without. Optimization is expressed in percentage of size or virtual-size economized. The Script Hash (SH) consumes a multi-signature script, and the Public Key Hash (PKH) consumes a standard public key. Note that size can vary by a few bytes with SegWit.

The average fee per virtual byte in the last three months was around 292 Satoshis. This optimization allows savings of up to 32,412 Satoshis for the first refund transaction without SegWit, and 12,264 Satoshis for a refund or a settlement transaction with SegWit. At the current price, these savings represent between USD \$1.31 and USD \$3.47[2]. If the channel is used for micropayments such as a couple of cents each time, this optimization makes a difference and lowers the required threshold for feasibility. The first refund transaction being less expensive also makes the clients commitment easier. The Table 1 exhibits transaction utilizing only one input, and it is worth noting that the number of input has a supralinear influence to the savings.

Requirements need to be defined to be able to substitute the multi-signature script with a threshold scheme. Analysis of the protocol and the signing process for a multi-signature script allows one to define these requirements. A 2-out-of-2 multi-signature script can be unlocked with two different public keys and their signature. The signing order only matters in that it is determined at the time of creating a multisig address. Some standards such as BIP45 address the need to predefine or communicate the ordering of the keys (and therefore the signatures) by always ordering the keys lexicographically, and always ordering the signatures in order of the keys [1]. The protocol takes advantage of this fact. A transaction is usually held fully signed only by one player. The threshold scheme must follow these requirements (i) 2 players need to co-operate to generate a valid signature,

[1] The transaction size is calculated with nested-SegWit and not with native mode
[2] Average price of Bitcoin in the last 3 months, around \$10,700 USD

		Non-SegWit		SegWit		
		R-Size	O	R-Size	V-Size	O
First Refund	SH	302	36.75%	340	174	22.99%
	PKH	191		216	134	
Refund Normal	SH	335	32.54%	372	207	20.29%
	PKH	226		246	165	
Settlement	SH	335	32.54%	372	207	20.29%
	PKH	226		246	165	

Table 1. Summary of transaction size optimization

(ii) both must be able to start the signing process, and (iii) only one player must be able to retrieve the signature at the end of the process. If both need the signature it is always feasible to share, meaning the current protocol is not better in this case.

To achieve this, 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 [7].

5.2 Pre-authorized Payments

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

Further research can be done in this area to figure out the feasibility and the most effective way to implement this feature in this scheme. Maybe a third layer, on top of layer two, is necessary and achievable, maybe the channel dimension can be increased.

6 Related Work

Simple micropayment channels were introduced by Hearn and Spilman [4]. The Lightning Network by Poon and Dryja [9], 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 [3].

7 Conclusion

Trustless one-way payment channels for Bitcoin resolve many problems. Scalability is near infinite and costs of the channel decrease linearly with the number of transactions in 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 in the channel and can withdraw arbitrary amounts and refill the channel at any time. The provider does not need to lock funds to receive money and has no cost to setup a channel with a client.

8 Acknowledgement

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

References

[1] Manuel Araoz, Ryan X. Charles, and Matias Alejo Garcia. *Structure for Deterministic P2SH Multisignature Wallets*. 2014. URL: `https://github.com/bitcoin/bips/blob/master/bip-0045.mediawiki` (visited on 02/09/2018).

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

[3] 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.

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

[5] Eric Lombrozo, Johnson Lau, and Pieter Wuille. *Segregated Witness*. URL: `https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki` (visited on 02/02/2018).

[6] Dr. Mark B Lundeberg. *Secret size attack on cross-chain hash lock smart contracts*. Feb. 15, 2018. URL: `https://gist.github.com/markblundeberg/7a932c98179de2190049f5823907c016`.

[7] 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.

[8] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: `http://bitcoin.org/bitcoin.pdf`.

[9] Joseph Poon and Thaddeus Dryja. "The bitcoin lightning network: Scalable off-chain instant payments". In: *draft version 0.5* 9 (2016).

[10] *SegWit*. URL: https://en.wikipedia.org/wiki/SegWit (visited on 02/02/2018).
[11] *SegWit2x*. URL: https://en.wikipedia.org/wiki/SegWit2x (visited on 02/08/2018).