The Bitcoin Lightning Network:
Scalable Off-Chain Instant Payments

Joseph Poon
joseph@lightning.network

Thaddeus Dryja
rx@awsomnet.org

January 14, 2016
DRAFT Version 0.5.9.2

Abstract

The bitcoin protocol can encompass the global financial transaction volume in all electronic payment systems today, without a single custodial third party holding funds or requiring participants to have anything more than a computer using a broadband connection. A decentralized system is proposed whereby transactions are sent over a network of micropayment channels (a.k.a. payment channels or transaction channels) whose transfer of value occurs off-blockchain. If Bitcoin transactions can be signed with a new sighash type that addresses malleability, these transfers may occur between untrusted parties along the transfer route by contracts which, in the event of uncooperative or hostile participants, are enforceable via broadcast over the bitcoin blockchain in the event of uncooperative or hostile participants, through a series of decrementing timelocks.

1 The Bitcoin Blockchain Scalability Problem

The Bitcoin[1] blockchain holds great promise for distributed ledgers, but the blockchain as a payment platform, by itself, cannot cover the world's commerce anytime in the near future. The blockchain is a gossip protocol whereby all state modifications to the ledger are broadcast to all participants. It is through this "gossip protocol" that consensus of the state, everyone's balances, is agreed upon. If each node in the bitcoin network must know about every single transaction that occurs globally, that may create a significant drag on the ability of the network to encompass all global financial transactions. It would instead be desirable to encompass all transactions in a way that doesn't sacrifice the decentralization and security that the network provides.

The payment network Visa achieved 47,000 peak transactions per second (tps) on its network during the 2013 holidays[2], and currently averages hundreds of millions per day. Currently, Bitcoin supports less than 7 transactions per second with a 1 megabyte block limit. If we use an average of 300 bytes per bitcoin transaction and assumed unlimited block sizes, an equivalent capacity to peak Visa transaction volume of 47,000/tps would be nearly 8 gigabytes per Bitcoin block, every ten minutes on average. Continuously, that would be over 400 terabytes of data per year.

Clearly, achieving Visa-like capacity on the Bitcoin network isn't feasible today. No home computer in the world can operate with that kind of bandwidth and storage. If Bitcoin is to replace all electronic payments in the future, and not just Visa, it would result in outright collapse of the Bitcoin network, or at best, extreme centralization of Bitcoin nodes and miners to the only ones who could afford it. This centralization would then defeat aspects of network decentralization that make Bitcoin secure, as the ability for entities to validate the chain is what allows Bitcoin to ensure ledger accuracy and security.

Having fewer validators due to larger blocks not only implies fewer individuals ensuring ledger accuracy, but also results in fewer entities that would be able to validate the blockchain as part of the mining process, which results in encouraging miner centralization. Extremely large blocks, for example in the above case of 8 gigabytes every 10 minutes on average, would imply that only a few parties would be able to do block validation. This creates a great possibility that entities will end up trusting centralized parties. Having privileged, trusted parties creates a social trap whereby the central party will not act in the interest of an individual (principal-agent problem), e.g. rentierism by charging higher fees to mitigate the incentive to act dishonestly. In extreme cases, this manifests as individuals sending funds to centralized trusted custodians who have full custody of customers' funds. Such arrangements, as are common today, create severe counterparty risk. A prerequisite to prevent that kind of centralization from occurring would require the ability for bitcoin to be validated by a single consumer-level computer on a home broadband connection. By ensuring that full validation can occur cheaply, Bitcoin nodes and miners will be able to prevent extreme centralization and trust, which ensures extremely low transaction fees.

While it is possible that Moore's Law will continue indefinitely, and the computational capacity for nodes to cost-effectively compute multi-gigabyte blocks may exist in the future, it is not a certainty.

To achieve much higher than 47,000 transactions per second using Bitcoin requires conducting transactions off the Bitcoin blockchain itself. It would be even better if the bitcoin network supported a near-unlimited number of transactions per second with extremely low fees for micropayments. Many micropayments can be sent sequentially between two parties to enable any size of payments. Micropayments would enable unbunding, less trust and commodification of services, such as payments for per-megabyte internet service. To be able to achieve these micropayment use cases, however, would require severely reducing the amount of transactions that end up being broadcast on the global Bitcoin blockchain.

While it is possible to scale at a small level, it is absolutely not possible to handle a large amount of micropayments on the network or to encompass all global transactions. For bitcoin to succeed, it requires confidence that if it were to become extremely popular, its current advantages stemming from decentralization will continue to exist. In order for people today to believe that Bitcoin will work tomorrow, Bitcoin needs to resolve the issue of block size centralization effects; large blocks implicitly create trusted custodians and significantly higher fees.

2 A Network of Micropayment Channels Can Solve Scalability

> "If a tree falls in the forest and no one is around to hear it, does it make a sound?"

The above quote questions the relevance of unobserved events —if nobody hears the tree fall, whether it made a sound or not is of no consequence. Similarly, in the blockchain, if only two participants care about an everyday recurring transaction, it's not necessary for all other nodes in the bitcoin network to know about that transaction. It is instead preferable to only have the bare minimum of information on the blockchain. By deferring telling the entire world about every transaction, doing net settlement of their relationship at a later date enables Bitcoin users to conduct many transactions without bloating up the blockchain or creating trust in a centralized counterparty. An effectively trustless structure can be achieved by using time locks as a component to global consensus.

Currently the solution to micropayments and scalability is to offload the transactions to a custodian, whereby one is trusting third party custodians to hold one's coins and to update balances with other parties. Trusting third parties to hold all of one's funds creates counterparty risk and transaction costs.

Instead, using a network of these micropayment channels, Bitcoin can scale to billions of transactions per day with the computational power available on a modern desktop computer today. Sending many payments inside a given micropayment channel enables one to send large amounts of funds to another party in a decentralized manner. These channels are not a separate trusted network on top of bitcoin. They are real bitcoin transactions.

Micropayment channels[3][4] create a relationship between two parties to perpetually update balances, deferring what is broadcast to the blockchain in a single transaction netting out the total balance between those two parties. This permits the financial relationships between two parties to be trustlessly deferred to a later date, without risk of counterparty default. Micropayment channels use real bitcoin transactions, only electing to defer the broadcast to the blockchain in such a way that both parties can guarantee their current balance on the blockchain; this is not a trusted overlay network —payments in micropayment channels are real bitcoin communicated and exchanged off-chain.

2.1 Micropayment Channels Do Not Require Trust

Like the age-old question of whether the tree falling in the woods makes a sound, if all parties agree that the tree fell at 2:45 in the afternoon, then the tree really did fall at 2:45 in the afternoon. Similarly, if both counterparties agree that the current balance inside a channel is 0.07 BTC to Alice and 0.03

BTC to Bob, then that's the true balance. However, without cryptography, an interesting problem is created: If one's counterparty disagrees about the current balance of funds (or time the tree fell), then it is one's word against another. Without cryptographic signatures, the blockchain will not know who owns what.

If the balance in the channel is 0.05 BTC to Alice and 0.05 BTC to Bob, and the balance after a transaction is 0.07 BTC to Alice and 0.03 BTC to Bob, the network needs to know which set of balances is correct. Blockchain transactions solve this problem by using the blockchain ledger as a timestamping system. At the same time, it is desirable to create a system which does not actively use this timestamping system unless absolutely necessary, as it can become costly to the network.

Instead, both parties can commit to signing a transaction and not broadcasting this transaction. So if Alice and Bob commit funds into a 2-of-2 multisignature address (where it requires consent from both parties to create spends), they can agree on the current balance state. Alice and Bob can agree to create a refund from that 2-of-2 transaction to themselves, 0.05 BTC to each. This refund is *not* broadcast on the blockchain. Either party may do so, but they may elect to instead hold onto that transaction, knowing that they are able to redeem funds whenever they feel comfortable doing so. By deferring broadcast of this transaction, they may elect to change this balance at a future date.

To update the balance, both parties create a new spend from the 2-of-2 multisignature address, for example 0.07 to Alice and 0.03 to Bob. Without proper design, though, there is the timestamping problem of not knowing which spend is correct: the new spend or the original refund.

The restriction on timestamping and dates, however, is not as complex as full ordering of all transactions as in the bitcoin blockchain. In the case of micropayment channels, only two states are required: the current correct balance, and any old deprecated balances. There would only be a single correct current balance, and possibly many old balances which are deprecated.

Therefore, it is possible in bitcoin to devise a bitcoin script whereby all old transactions are invalidated, and only the new transaction is valid. Invalidation is enforced by a bitcoin output script and dependent transactions which force the other party to give all their funds to the channel counterparty. By taking all funds as a penalty to give to the other, all old transactions are thereby invalidated.

This invalidation process can exist through a process of channel consensus where if both parties agree on current ledger states (and building new states), then the real balance gets updated. The balance is reflected on the blockchain only when a single party disagrees. Conceptually, this system is not an independent overlay network; it is more a deferral of state on the current system, as the enforcement is still occurring on the blockchain itself (albeit deferred to future dates and transactions).

2.2 A Network of Channels

Thus, micropayment channels only create a relationship between two parties. Requiring everyone to create channels with everyone else does not solve the scalability problem. Bitcoin scalability can be achieved using a large network of micropayment channels.

If we presume a large network of channels on the Bitcoin blockchain, and all Bitcoin users are participating on this graph by having at least one channel open on the Bitcoin blockchain, it is possible to create a near-infinite amount of transactions inside this network. The only transactions that are broadcasted on the Bitcoin blockchain prematurely are with uncooperative channel counterparties.

By encumbering the Bitcoin transaction outputs with a hashlock and timelock, the channel counterparty will be unable to outright steal funds and Bitcoins can be exchanged without outright counterparty theft. Further, by using staggered timeouts, it's possible to send funds via multiple intermediaries in a network without the risk of intermediary theft of funds.

3 Bidirectional Payment Channels

Micropayment channels permit a simple deferral of a transaction state to be broadcast at a later time. The contracts are enforced by creating a responsibility for one party to broadcast transactions before or after certain dates. If the blockchain is a decentralized timestamping system, it is possible to use clocks as a component of decentralized consensus[5] to determine data validity, as well as present states as a method to order events[6].

By creating timeframes where certain states can be broadcast and later invalidated, it is possible to create complex contracts using bitcoin transaction scripts. There has been prior work for Hub-and-Spoke Micropayment Channels[7][8][9] (and trusted payment channel networks[10][11]) looking at building a hub-and-spoke network today. However, Lightning Network's bidirectional micropayment channel requires the malleability softfork described in Appendix A to enable near-infinite scalability while mitigating risks of intermediate node default.

By chaining together multiple micropayment channels, it is possible to create a network of transaction paths. Paths can be routed using a BGP-like system, and the sender may designate a particular path to the recipient. The output scripts are encumbered by a hash, which is generated by the recipient. By disclosing the input to that hash, the recipient's counterparty will be able to pull funds along the route.

3.1 The Problem of Blame in Channel Creation

In order to participate in this payment network, one must create a micropayment channel with another participant on this network.

3.1.1 Creating an Unsigned Funding Transaction

An initial channel Funding Transaction is created whereby one or both channel counterparties fund the inputs of this transaction. Both parties create the inputs and outputs for this transaction but do not sign the transaction.

The output for this Funding Transaction is a single 2-of-2 multisignature script with both participants in this channel, henceforth named Alice and Bob. Both participants do not exchange signatures for the Funding Transaction until they have created spends from this 2-of-2 output refunding the original amount back to its respective funders. The purpose of not signing the transaction allows for one to spend from a transaction which does not yet exist. If Alice and Bob exchange the signatures from the Funding Transaction without being able to broadcast spends from the Funding Transaction, the funds may be locked up forever if Alice and Bob do not cooperate (or other coin loss may occur through hostage scenarios whereby one pays for the cooperation from the counterparty).

Alice and Bob both exchange inputs to fund the Funding Transaction (to know which inputs are used to determine the total value of the channel), and exchange one key to use to sign with later. This key is used for the 2-of-2 output for the Funding Transaction; both signatures are needed to spend from the Funding Transaction, in other words, both Alice and Bob need to agree to spend from the Funding Transaction.

3.1.2 Spending from an Unsigned Transaction

The Lightning Network uses a SIGHASH_NOINPUT transaction to spend from this 2-of-2 Funding Transaction output, as it is necessary to spend from a transaction for which the signatures are not yet exchanged. SIGHASH_NOINPUT, implemented using a soft-fork, ensures transactions can be spent from before it is signed by all parties, as transactions would need to be signed to get a transaction ID without new sighash flags. Without SIGHASH_NOINPUT, Bitcoin transactions cannot be spent from before they may be broadcast —it's as if one could not draft a contract without paying the other party first. SIGHASH_NOINPUT resolves this problem. See Appendix A for more information and implementation.

Without SIGHASH_NOINPUT, it is not possible to generate a spend from a transaction without exchanging signatures, since spending the Funding Transaction requires a transaction ID as part of the signature in the child's input. A component of the Transaction ID is the parent's (Funding Transaction's) signature, so both parties need to exchange their signatures of the parent transaction before the child can be spent. Since one or both parties must know the parent's signatures to spend from it, that means one or both parties are able to broadcast the parent (Funding Transaction) before the child even exists. SIGHASH_NOINPUT gets around this by permitting the child to spend without signing the input. With SIGHASH_NOINPUT, the order of operations are to:

1. Create the parent (Funding Transaction)
2. Create the children (Commitment Transactions and all spends from the commitment transactions)
3. Sign the children
4. Exchange the signatures for the children

5. Sign the parent

6. Exchange the signatures for the parent

7. Broadcast the parent on the blockchain

One is not able to broadcast the parent (Step 7) until Step 6 is complete. Both parties have not given their signature to spend from the Funding Transaction until step 6. Further, if one party fails during Step 6, the parent can either be spent to become the parent transaction or the inputs to the parent transaction can be double-spent (so that this entire transaction path is invalidated).

3.1.3 Commitment Transactions: Unenforcible Construction

After the unsigned (and unbroadcasted) Funding Transaction has been created, both parties sign and exchange an initial Commitment Transaction. These Commitment Transactions spends from the 2-of-2 output of the Funding Transaction (parent). However, only the Funding Transaction is broadcast on the blockchain.

Since the Funding Transaction has already entered into the blockchain, and the output is a 2-of-2 multisignature transaction which requires the agreement of both parties to spend from, Commitment Transactions are used to express the present balance. If only one 2-of-2 signed Commitment Transaction is exchanged between both parties, then both parties will be sure that they are able to get their money back after the Funding Transaction enters the blockchain. Both parties do not broadcast the Commitment Transactions onto the blockchain until they want to close out the current balance in the channel. They do so by broadcasting the present Commitment Transaction.

Commitment Transactions pay out the respective current balances to each party. A naive (broken) implementation would construct an unbroadcasted transaction whereby there is a 2-of-2 spend from a single transaction which have two outputs that return all current balances to both channel counterparties. This will return all funds to the original party when creating an initial Commitment Transaction.

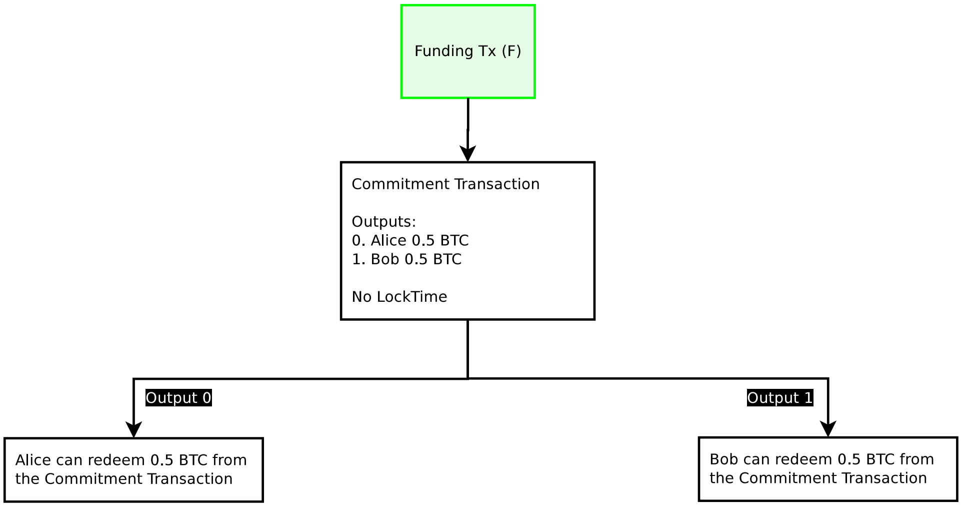

Funding Tx (F)
Commitment Transaction
Outputs:
0. Alice 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Output 0
Output 1
Alice can redeem 0.5 BTC from the Commitment Transaction
Bob can redeem 0.5 BTC from the Commitment Transaction

Figure 1: A naive broken funding transaction is described in this diagram. The Funding Transaction (F), designated in green, is broadcast on the blockchain after all other transactions are signed. All other transactions spending from the funding transactions are not yet broadcast, in case the counterparties wish to update their balance. Only the Funding Transaction is broadcast on the blockchain at this time.

For instance, if Alice and Bob agree to create a Funding Transaction with a single 2-of-2 output worth 1.0 BTC (with 0.5 BTC contribution from each), they create a Commitment Transaction where there are two 0.5 BTC outputs for Alice and Bob. The Commitment Transactions are signed first and keys are exchanged so either is able to broadcast the Commitment Transaction at any time contingent upon the Funding Transaction entering into the blockchain. At this point, the Funding Transaction signatures can safely be exchanged, as either party is able to redeem their funds by broadcasting the Commitment Transaction.

This construction breaks, however, when one wishes to update the present balance. In order to update the balance, they must update their Commitment Transaction output values (the Funding Transaction has already entered into the blockchain and cannot be changed).

When both parties agree to a new Commitment Transaction and exchange signatures for the new Commitment Transaction, either Commitment Transactions can be broadcast. As the output from the Funding Transaction can only be redeemed once, only one of those transactions will be valid. For instance, if Alice and Bob agree that the balance of the channel is now 0.4 to Alice and 0.6 to Bob, and a new Commitment Transaction is created to reflect that, either Commitment Transaction can be broadcast. In effect, one would be unable to restrict which Commitment Transaction is broadcast, since both parties have signed and exchanged the signatures for either balance to be broadcast.

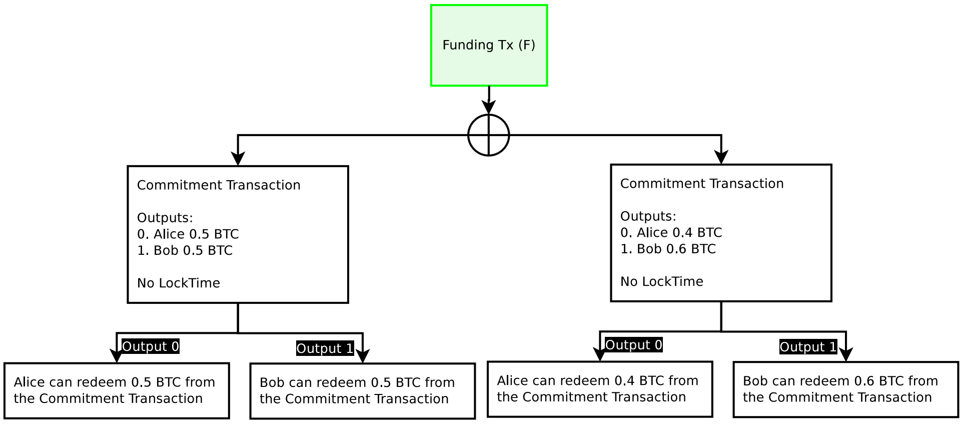

Funding Tx (F)
Commitment Transaction
Outputs:
0. Alice 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Commitment Transaction
Outputs:
0. Alice 0.4 BTC
1. Bob 0.6 BTC
No LockTime
Output 0
Output 1
Output 0
Output 1
Alice can redeem 0.5 BTC from the Commitment Transaction
Bob can redeem 0.5 BTC from the Commitment Transaction
Alice can redeem 0.4 BTC from the Commitment Transaction
Bob can redeem 0.6 BTC from the Commitment Transaction

Figure 2: Either of the Commitment Transactions can be broadcast any any time by either party, only one will successfully spend from the single Funding Transaction. This cannot work because one party will not want to broadcast the most recent transaction.

Since either party may broadcast the Commitment Transaction at any time, the result would be after the new Commitment Transaction is generated, the one who receives less funds has significant incentive to broadcast the transaction which has greater values for themselves in the Commitment Transaction outputs. As a result, the channel would be immediately closed and funds stolen. Therefore, one cannot create payment channels under this model.

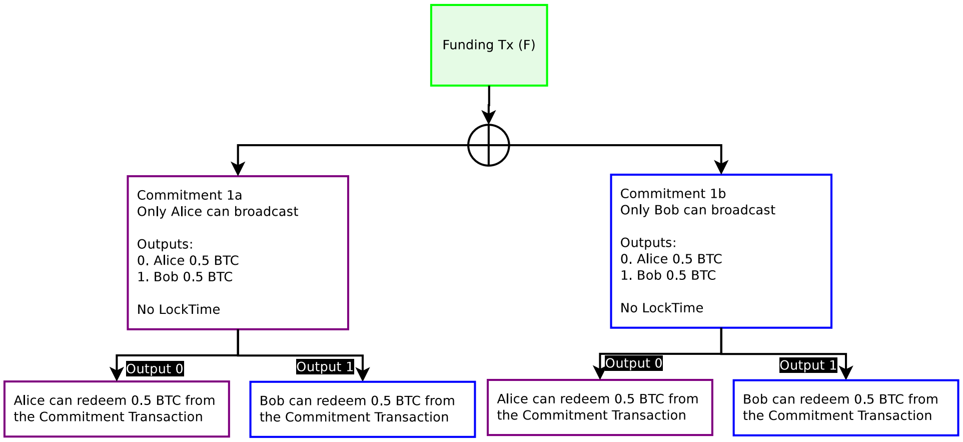
3.1.4 Commitment Transactions: Ascribing Blame

Since any signed Commitment Transaction may be broadcast on the blockchain, and only one can be successfully broadcast, it is necessary to prevent old Commitment Transactions from being broadcast. It is not possible to revoke tens of thousands of transactions in Bitcoin, so an alternate method is necessary. Instead of active revocation enforced by the blockchain, it's necessary to construct the channel itself in similar manner to a Fidelity Bond, whereby both parties make commitments, and violations of these commitments are enforced by penalties. If one party violates their agreement, then they will lose all the money in the channel.

For this payment channel, the contract terms are that both parties commit to broadcasting only the most recent transaction. Any broadcast of older transactions will cause a violation of the contract, and all funds are given to the other party as a penalty.

This can only be enforced if one is able to ascribe blame for broadcasting an old transaction. In order to do so, one must be able to uniquely identify who broadcast an older transaction. This can be done if each counterparty has a uniquely identifiable Commitment Transaction. Both parties must sign the inputs to the Commitment Transaction which the other party is responsible for broadcasting. Since one has a version of the Commitment Transaction that is signed by the other party, one can only broadcast one's own version of the Commitment Transaction.

For the Lightning Network, all spends from the Funding Transaction output, Commitment Transactions, have two half-signed transactions. One Commitment Transaction in which Alice signs and gives to Bob (C1b), and another which Bob signs and gives to Alice (C1a). These two Commitment Transactions spend from the same output (Funding Transaction), and have different contents; only one can be broadcast on the blockchain, as both pairs of Commitment Transactions spend from the same Funding Transaction. Either party may broadcast their received Commitment Transaction by signing their version and including the counterparty's signature. For example, Bob can broadcast Commitment C1b, since he has already received the signature for C1b from Alice —he includes Alice's signature and signs C1b himself. The transaction will be a valid spend from the Funding Transaction's 2-of-2 output requiring both Alice and Bob's signature.



Funding Tx (F)
Commitment 1a
Only Alice can broadcast
Outputs:
0. Alice 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Commitment 1b
Only Bob can broadcast
Outputs:
0. Alice 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Output 0
Output 1
Output 0
Output 1
Alice can redeem 0.5 BTC from the Commitment Transaction
Bob can redeem 0.5 BTC from the Commitment Transaction
Alice can redeem 0.5 BTC from the Commitment Transaction
Bob can redeem 0.5 BTC from the Commitment Transaction

Figure 3: Purple boxes are unbroadcasted transactions which only Alice can broadcast. Blue boxes are unbroadcasted transaction which only Bob can broadcast. Alice can only broadcast Commitment 1a, Bob can only broadcast Commitment 1b. Only one Commitment Transaction can be spent from the Funding Transaction output. Blame is ascribed, but either one can still be spent with no penalty.

However, even with this construction, one has only merely allocated blame. It is not yet possible to enforce this contract on the Bitcoin blockchain. Bob still trusts Alice not to broadcast an old Commitment Transaction. At this time, he is only able to prove that Alice has done so via a half-signed transaction proof.

3.2 Creating a Channel with Contract Revocation

To be able to actually enforce the terms of the contract, it's necessary to construct a Commitment Transaction (along with its spends) where one is able to revoke a transaction. This revocation is achievable by using data about when a transaction enters into a blockchain and using the maturity of the transaction to determine validation paths.

3.3 Sequence Number Maturity

Mark Freidenbach has proposed that Sequence Numbers can be enforcible via a relative block maturity of the parent transaction via a soft-fork[12]. This would allow some basic ability to ensure some form of relative block confirmation time lock on the spending script. In addition, an additional opcode, OP_CHECKSEQUENCEVERIFY[13] (a.k.a. OP_RELATIVECHECKLOCKTIMEVERIFY)[14], would permit further abilities, including allowing a stop-gap solution before a more permanent solution for resolving transaction malleability. A future version of this paper will include proposed solutions.

To summarize, Bitcoin was released with a sequence number which was only enforced in the mempool of unconfirmed transactions. The original behavior permitted transaction replacement by replacing transactions in the mempool with newer transactions if they have a higher sequence number. Due to transaction replacement rules, it is not enforced due to denial of service attack risks. It appears as though the intended purpose of the sequence number is to replace unbroadcasted transactions. However, this higher sequence number replacement behavior is unenforcible. One cannot be assured that old versions of transactions were replaced in the mempool and a block contains the most recent version of the transaction. A way to enforce transaction versions off-chain is via time commitments.

A Revocable Transaction spends from a unique output where the transaction has a unique type of output script. This parent's output has two redemption paths where the first can be redeemed immediately, and the second can only be redeemed if the child has a minimum number of confirmations between transactions. This is achieved by making the sequence number of the child transaction require a minimum number of confirmations from the parent. In essence, this new sequence number behavior will only permit a spend from this output to be valid if the number of blocks between the output and the redeeming transaction is above a specified block height.

A transaction can be revoked with this sequence number behavior by creating a restriction with some defined number of blocks defined in the sequence number, which will result in the spend being only valid after the parent has entered into the blockchain for some defined number of blocks. This creates a structure whereby the parent transaction with this output becomes a bonded deposit, attesting that there is no revocation. A time period exists which anyone on the blockchain can refute this attestation by broadcasting a spend immediately after the transaction is broadcast.

If one wishes to permit revocable transactions with a 1000-confirmation delay, the output transaction construction would remain a 2-of-2 multisig:

```
2 <Alice1> <Bob1> 2 OP_CHECKMULTISIG
```

However, the child spending transaction would contain a nSequence value of 1000. Since this transaction requires the signature of both counterparties to be valid, both parties include the nSequence number of 1000 as part of the signature. Both parties may, at their discretion, agree to create another transaction which supersedes that transaction without any nSequence number.

This construction, a Revocable Sequence Maturity Contract (RSMC), creates two paths, with very specific contract terms.

The contract terms are:

1. All parties pay into a contract with an output enforcing this contract

2. Both parties may agree to send funds to some contract, with some waiting period (1000 confirmations in our example script). This is the revocable output balance.

3. One or both parties may elect to not broadcast (enforce) the payouts until some future date; either party may redeem the funds after the waiting period at any time.

4. If neither party has broadcast this transaction (redeemed the funds), they may revoke the above payouts if and only if both parties agree to do so by placing in a new payout term in a superseding transaction payout. The new transaction payout can be immediately redeemed after the contract is disclosed to the world (broadcast on the blockchain).

5. In the event that the contract is disclosed and the new payout structure is not redeemed, the prior revoked payout terms may be redeemed by either party (so it is the responsibility of either party to enforce the new terms).

The pre-signed child transaction can be redeemed after the parent transaction has entered into the blockchain with 1000 confirmations, due to the child's nSequence number on the input spending the parent.

In order to revoke this signed child transaction, both parties just agree to create another child transaction with the default field of the nSequence number of MAX_INT, which has special behavior permitting spending at any time.

This new signed spend supersedes the revocable spend so long as the new signed spend enters into the blockchain within 1000 confirmations of the parent transaction entering into the blockchain. In effect, if Alice and Bob agree to monitor the blockchain for incorrect broadcast of Commitment Transactions, the moment the transaction gets broadcast, they are able to spend using the superseding transaction immediately. In order to broadcast the revocable spend (deprecated transaction), which spends from the same output as the superseding transaction, they must wait 1000 confirmations. So long as both parties watch the blockchain, the revocable spend will never enter into the transaction if either party prefers the superseding transaction.

Using this construction, anyone could create a transaction, not broadcast the transaction, and then later create incentives to not ever broadcast that transaction in the future via penalties. This permits participants on the Bitcoin network to defer many transactions from ever hitting the blockchain.

3.3.1 Timestop

To mitigate a flood of transactions by a malicious attacker requires a credible threat that the attack will fail.

Greg Maxwell proposed using a timestop to mitigate a malicious flood on the blockchain:

> There are many ways to address this [flood risk] which haven't been adequately explored yet —for example, the clock can stop when blocks are full; turning the security risk into more hold-up delay in the event of a dos attack.[15]

This can be mitigated by allowing the miner to specify whether the current (fee paid) mempool is presently being flooded with transactions. They can enter a "1" value into the last bit in the version number of the block header. If the last bit in the block header contains a "1", then that block will not count towards the relative height maturity for the nSequence value and the block is designated as a congested block. There is an uncongested block height (which is always lower than the normal block height). This block height is used for the nSequence value, which only counts block maturity (confirmations).

A miner can elect to define the block as a congested block or not. The default code could automatically set the congested block flag as "1" if the mempool is above some size and the average fee for that set size is above some value. However, a miner has full discretion to change the rules on what automatically sets as a congested block, or can select to permanently set the congestion flag to be permanently on or off. It's expected that most honest miners would use the default behavior defined in their miner and not organize a 51% attack.

For example, if a parent transaction output is spent by a child with a nSequence value of 10, one must wait 10 confirmations before the transaction becomes valid. However, if the timestop flag has been set, the counting of confirmations stops, even with new blocks. If 6 confirmations have elapsed (4 more are necessary for the transaction to be valid), and the timestop block has been set on the 7th block, that block does not count towards the nSequence requirement of 10 confirmations; the child is still at 6 blocks for the relative confirmation value. Functionally, this will be stored as some kind of auxiliary timestop block height which is used only for tracking the timestop value. When the timestop bit is set, all transactions using an nSequence value will stop counting until the timestop bit has been unset. This gives sufficient time and block-space for transactions at the current auxiliary timestop block height to enter into the blockchain, which can prevent systemic attackers from successfully attacking the system.

However, this requires some kind of flag in the block to designate whether it is a timestop block. For full SPV compatibility (Simple Payment Verification; lightweight clients), it is desirable for this to be within the 80-byte block header instead of in the coinbase. There are two places which may be a good place to put in this flag in the block header: in the block time and in the block version. The block time may not be safe due to the last bits being used as an entropy source for some ASIC miners, therefore a bit may need to be consumed for timestop flags. Another option would be to hardcode timestop activation as a hard consensus rule (e.g. via block size), however this may make things less flexible. By setting sane defaults for timestop rules, these rules can be changed without consensus soft-forks.

If the block version is used as a flag, the contextual information must match the Chain ID used in some merge-mined coins.

3.3.2 Revocable Commitment Transactions

By combining the ascribing of blame as well as the revocable transaction, one is able to determine when a party is not abiding by the terms of the contract, and enforce penalties without trusting the counterparty.

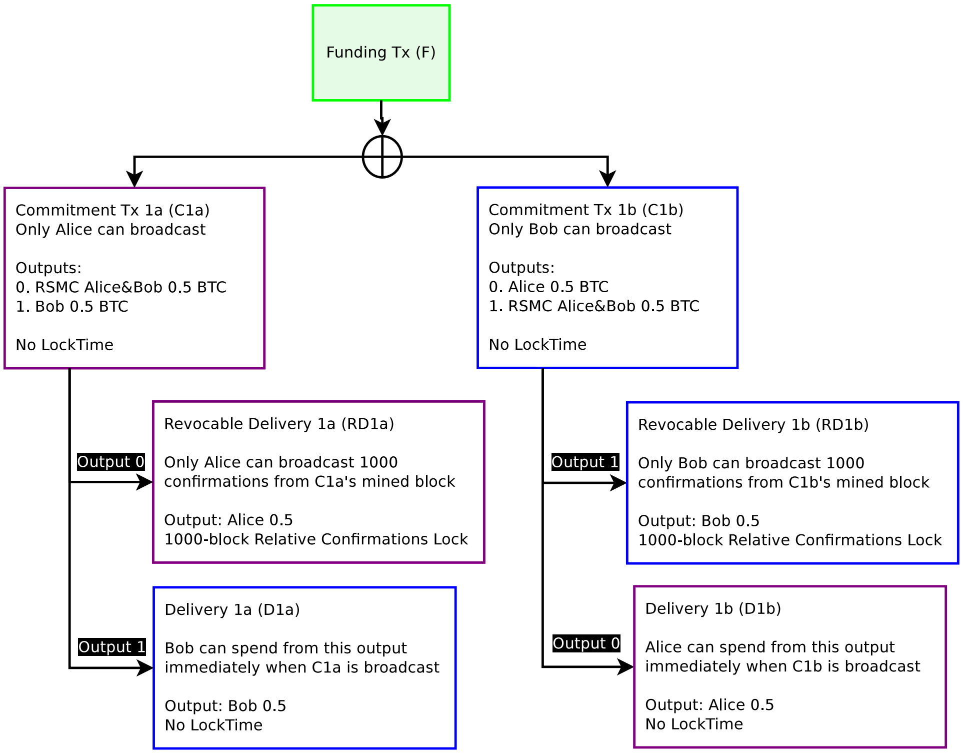

Funding Tx (F)
Commitment Tx 1a (C1a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Commitment Tx 1b (C1b)
Only Bob can broadcast
Outputs:
0. Alice 0.5 BTC
1. RSMC Alice&Bob 0.5 BTC
No LockTime
Output 0
Revocable Delivery 1a (RD1a)
Only Alice can broadcast 1000 confirmations from C1a's mined block
Output: Alice 0.5
1000-block Relative Confirmations Lock
Output 1
Revocable Delivery 1b (RD1b)
Only Bob can broadcast 1000 confirmations from C1b's mined block
Output: Bob 0.5
1000-block Relative Confirmations Lock
Output 1
Delivery 1a (D1a)
Bob can spend from this output immediately when C1a is broadcast
Output: Bob 0.5
No LockTime
Output 0
Delivery 1b (D1b)
Alice can spend from this output immediately when C1b is broadcast
Output: Alice 0.5
No LockTime

Figure 4: The Funding Transaction F, designated in green, is broadcast on the blockchain after all other transactions are signed. All transactions which only Alice can broadcast are in purple. All transactions which only Bob can broadcast is are blue. Only the Funding Transaction is broadcast on the blockchain at this time.

The intent of creating a new Commitment Transaction is to invalidate all old Commitment Transactions when updating the new balance with a new Commitment Transaction. Invalidation of old transactions can happen by making an output be a Revocable Sequence Maturity Contract (RSMC). To invalidate a transaction, a superseding transaction will be signed and exchanged by both parties that gives all funds to the counterparty in the event an older transaction is incorrectly broadcast. The incorrect broadcast is identified by creating two different Commitment Transactions with the same final balance outputs, however the payment to oneself is encumbered by an RSMC.

In effect, there are two Commitment Transactions from a single Funding Transaction 2-of-2 outputs. Of these two Commitment Transactions, only one can enter into the blockchain. Each party within a channel has one version of this contract. So if this is the first Commitment Transaction pair, Alice's Commitment Transaction is defined as C1a, and Bob's Commitment Transaction is defined as C1b. By broadcasting a Commitment Transaction, one is requesting for the channel to close out and end. The first two outputs for the Commitment Transaction include a Delivery Transaction (payout) of the present unallocated balance to the channel counterparties. If Alice broadcasts C1a, one of the output is spendable by D1a, which sends funds to Bob. For Bob, C1b is spendable by D1b, which sends funds to Alice. The Delivery Transaction (D1a/D1b) is immediately redeemable and is not encumbered in any way in the event the Commitment Transaction is broadcast.

For each party's Commitment Transaction, they are attesting that they are broadcasting the most recent Commitment Transaction which they own. Since they are attesting that this is the current balance, the balance paid to the counterparty is assumed to be true, since one has no direct benefit by paying some funds to the counterparty as a penalty.

The balance paid to the person who broadcast the Commitment Transaction, however, is unverified. The participants on the blockchain have no idea if the Commitment Transaction is the most recent or not. If they do not broadcast their most recent version, they will be penalized by taking all the funds in the channel and giving it to the counterparty. Since their own funds are encumbered in their own RSMC, they will only be able to claim their funds after some set number of confirmations after the Commitment Transaction has been included in a block (in our example, 1000 confirmations). If they do broadcast their most recent Commitment Transaction, there should be no revocation transaction superseding the revocable transaction, so they will be able to receive their funds after some set amount of time (1000 confirmations).

By knowing who broadcast the Commitment Transaction and encumbering one's own payouts to be locked up for a predefined period of time, both parties will be able to revoke the Commitment Transaction in the future.

3.3.3 Redeeming Funds from the Channel: Cooperative Counterparties

Either party may redeem the funds from the channel. However, the party that broadcasts the Commitment Transaction must wait for the predefined number of confirmations described in the RSMC. The counterparty which did not broadcast the Commitment Transaction may redeem the funds immediately.

For example, if the Funding Transaction is committed with 1 BTC (half to each counterparty) and Bob broadcasts the most recent Commitment Transaction, C1b, he must wait 1000 confirmations to receive his 0.5 BTC, while Alice can spend 0.5 BTC. For Alice, this transaction is fully closed if Alice agrees that Bob broadcast the correct Commitment Transaction (C1b).

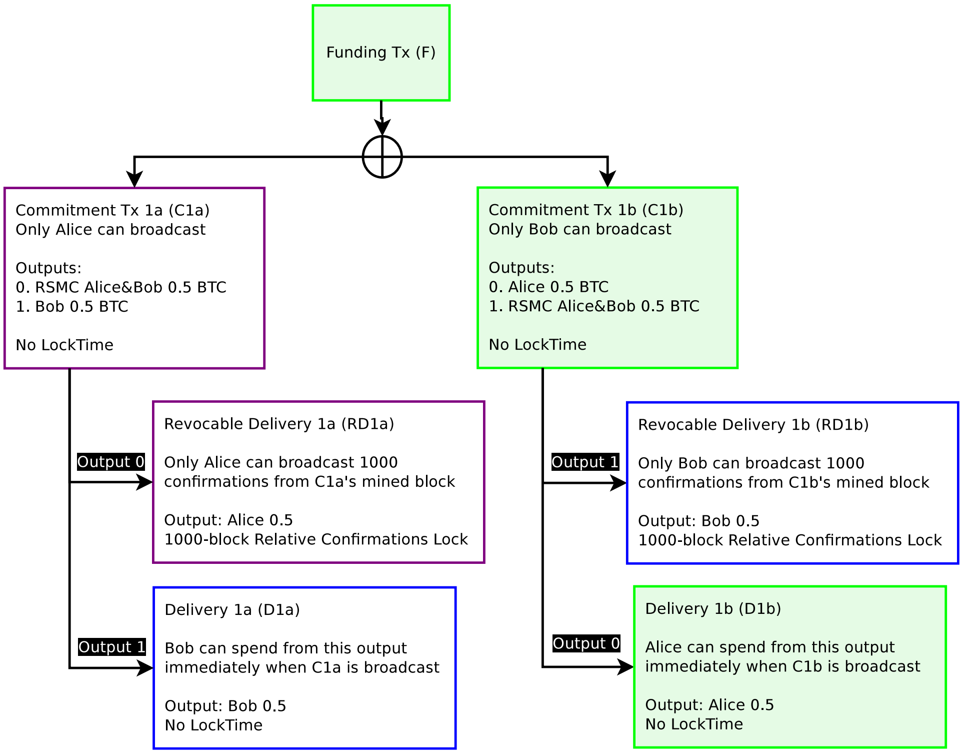

Funding Tx (F)
Commitment Tx 1a (C1a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Commitment Tx 1b (C1b)
Only Bob can broadcast
Outputs:
0. Alice 0.5 BTC
1. RSMC Alice&Bob 0.5 BTC
No LockTime
Output 0
Revocable Delivery 1a (RD1a)
Only Alice can broadcast 1000
confirmations from C1a's mined block
Output: Alice 0.5
1000-block Relative Confirmations Lock
Output 1
Revocable Delivery 1b (RD1b)
Only Bob can broadcast 1000
confirmations from C1b's mined block
Output: Bob 0.5
1000-block Relative Confirmations Lock
Output 1
Delivery 1a (D1a)
Bob can spend from this output
immediately when C1a is broadcast
Output: Bob 0.5
No LockTime
Output 0
Delivery 1b (D1b)
Alice can spend from this output
immediately when C1b is broadcast
Output: Alice 0.5
No LockTime

Figure 5: When Bob broadcasts C1b, Alice can immediately redeem her portion. Bob must wait 1000 confirmations. When the block is immediately broadcast, it is in this state. Transactions in green are transactions which are committed into the blockchain.

After the Commitment Transaction has been in the blockchain for 1000 blocks, Bob can then broadcast the Revocable Delivery transaction. He must wait 1000 blocks to prove he has not revoked this Commitment Transaction (C1b). After 1000 blocks, the Revocable Delivery transaction will be able to be included in a block. If a party attempt to include the Revocable Delivery transaction in a block before 1000 confirmations, the transaction will be invalid up until after 1000 confirmations have passed (at which point it will become valid if the output has not yet been redeemed).

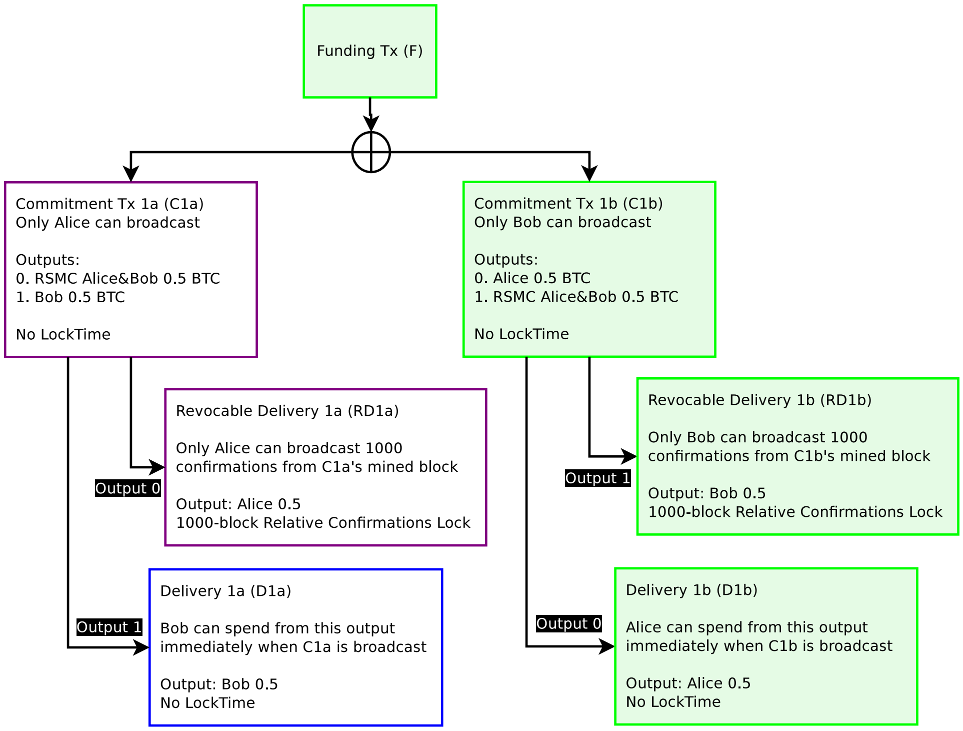

Funding Tx (F)
Commitment Tx 1a (C1a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Commitment Tx 1b (C1b)
Only Bob can broadcast
Outputs:
0. Alice 0.5 BTC
1. RSMC Alice&Bob 0.5 BTC
No LockTime
Output 0
Revocable Delivery 1a (RD1a)
Only Alice can broadcast 1000 confirmations from C1a's mined block
Output: Alice 0.5
1000-block Relative Confirmations Lock
Output 1
Revocable Delivery 1b (RD1b)
Only Bob can broadcast 1000 confirmations from C1b's mined block
Output: Bob 0.5
1000-block Relative Confirmations Lock
Output 1
Delivery 1a (D1a)
Bob can spend from this output immediately when C1a is broadcast
Output: Bob 0.5
No LockTime
Output 0
Delivery 1b (D1b)
Alice can spend from this output immediately when C1b is broadcast
Output: Alice 0.5
No LockTime

Figure 6: Alice agrees that Bob broadcast the correct Commitment Transaction and 1000 confirmations have passed. Bob then is able to broadcast the Revocable Delivery (RD1b) transaction on the blockchain.

After Bob broadcasts the Revocable Delivery transaction, the channel is fully closed for both Alice and Bob, everyone has received the funds which they both agree are the current balance they each own in the channel.

If it was instead Alice who broadcast the Commitment Transaction (C1a), she is the one who must wait 1000 confirmations instead of Bob.

3.3.4 Creating a new Commitment Transaction and Revoking Prior Commitments

While each party may close out the most recent Commitment Transaction at any time, they may also elect to create a new Commitment Transaction and invalidate the old one.

Suppose Alice and Bob now want to update their current balances from 0.5 BTC each refunded to 0.6 BTC for Bob and 0.4 BTC for Alice.

When they both agree to do so, they generate a new pair of Commitment Transactions.

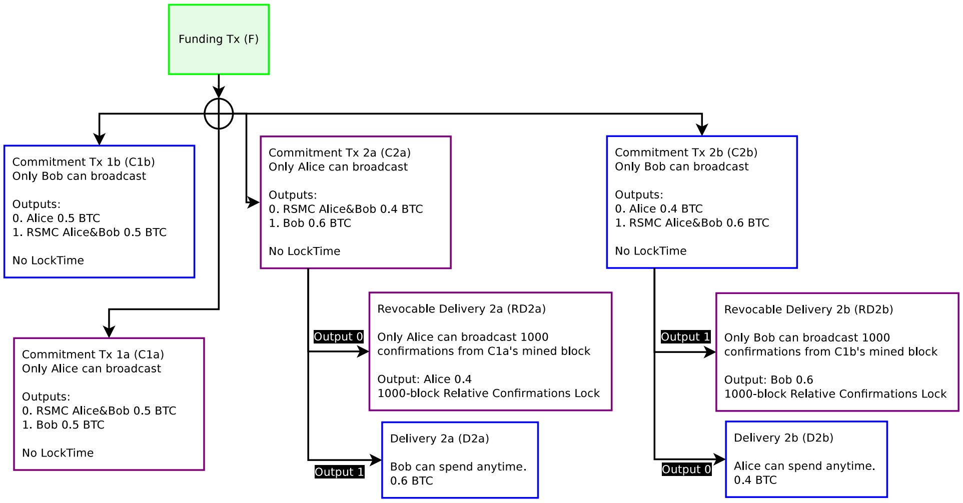


Figure 7: Four possible transactions can exist, a pair with the old commitments, and another pair with the new commitments. Each party inside the channel can only broadcast half of the total commitments (two each). There is no explicit enforcement preventing any particular Commitment being broadcast other than penalty spends, as they are all valid unbroadcasted spends. The Revocable Commitment still exists with the C1a/C1b pair, but are not displayed for brevity.

When a new pair of Commitment Transactions (C2a/C2b) is agreed upon, both parties will sign and exchange signatures for the new Commitment Transaction, then invalidate the old Commitment Transaction. This invalidation occurs by having both parties sign a Breach Remedy Transaction (BR1), which supersedes the Revocable Delivery Transaction (RD1). Each party hands to the other a half-signed revocation (BR1) from their own Revocable Delivery (RD1), which is a spend from the Commitment Transaction. The Breach Remedy Transaction will send all coins to the counterparty within the current balance of the channel. For example, if Alice and Bob both generate a new pair of Commitment Transactions (C2a/C2b) and invalidate prior commitments (C1a/C1b), and later Bob incorrectly broadcasts C1b on the blockchain, Alice can take all of Bob's money from the channel. Alice can do this because Bob has proved to Alice via penalty that he will never broadcast C1b, since the moment he broadcasts C1b, Alice is able to take all of Bob's money in the channel. In effect, by constructing a Breach

Remedy transaction for the counterparty, one has attested that one will not be broadcasting any prior commitments. The counterparty can accept this, because they will get all the money in the channel when this agreement is violated.

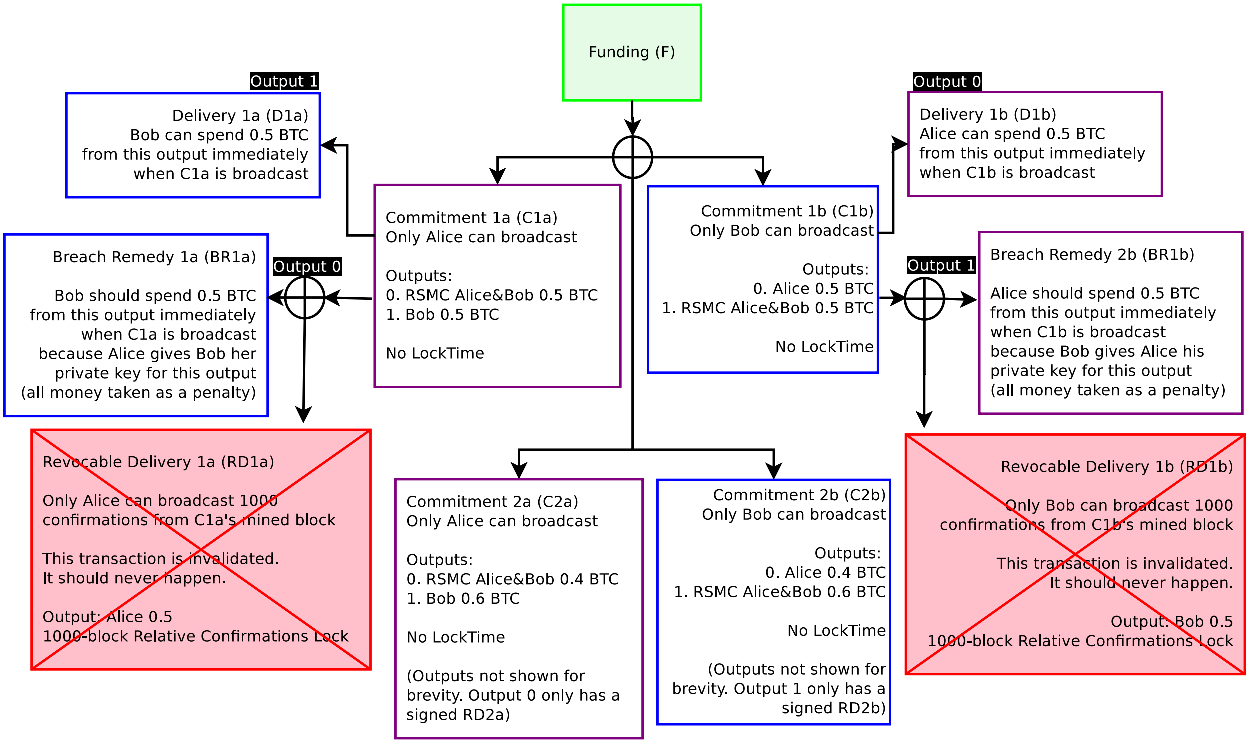

Funding (F)
Output 1
Delivery 1a (D1a)
Bob can spend 0.5 BTC
from this output immediately
when C1a is broadcast
Output 0
Delivery 1b (D1b)
Alice can spend 0.5 BTC
from this output immediately
when C1b is broadcast
Commitment 1a (C1a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Commitment 1b (C1b)
Only Bob can broadcast
Outputs:
0. Alice 0.5 BTC
1. RSMC Alice&Bob 0.5 BTC
No LockTime
Breach Remedy 1a (BR1a)
Bob should spend 0.5 BTC
from this output immediately
when C1a is broadcast
because Alice gives Bob her
private key for this output
(all money taken as a penalty)
Output 0
Output 1
Breach Remedy 2b (BR1b)
Alice should spend 0.5 BTC
from this output immediately
when C1b is broadcast
because Bob gives Alice his
private key for this output
(all money taken as a penalty)
Revocable Delivery 1a (RD1a)
Only Alice can broadcast 1000
confirmations from C1a's mined block
This transaction is invalidated.
It should never happen.
Output: Alice 0.5
1000-block Relative Confirmations Lock
Commitment 2a (C2a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.4 BTC
1. Bob 0.6 BTC
No LockTime
(Outputs not shown for
brevity. Output 0 only has a
signed RD2a)
Commitment 2b (C2b)
Only Bob can broadcast
Outputs:
0. Alice 0.4 BTC
1. RSMC Alice&Bob 0.6 BTC
No LockTime
(Outputs not shown for
brevity. Output 1 only has a
signed RD2b)
Revocable Delivery 1b (RD1b)
Only Bob can broadcast 1000
confirmations from C1b's mined block
This transaction is invalidated.
It should never happen.
Output: Bob 0.5
1000-block Relative Confirmations Lock

Figure 8: When C2a and C2b exist, both parties exchange Breach Remedy transactions. Both parties now have explicit economic incentive to avoid broadcasting old Commitment Transactions (C1a/C1b). If either party wishes to close out the channel, they will only use C2a (Alice) or C2b (Bob). If Alice broadcasts C1a, all her money will go to Bob. If Bob broadcasts C1b, all his money will go to Alice. See previous figure for C2a/C2b outputs.

Due to this fact, one will likely delete all prior Commitment Transactions when a Breach Remedy Transaction has been passed to the counterparty. If one broadcasts an incorrect (deprecated and invalidated Commitment Transaction), all the money will go to one's counterparty. For example, if Bob broadcasts C1b, so long as Alice watches the blockchain within the predefined number of blocks (in this case, 1000 blocks), Alice will be able to take all the money in this channel by broadcasting RD1b. Even if the present balance of the Commitment state (C2a/C2b) is 0.4 BTC to Alice and 0.6 BTC to Bob, because Bob violated the terms of the contract, all the money goes to Alice as a penalty. Functionally, the Revocable Transaction acts as a proof to the blockchain that Bob has violated the terms in the channel and this is programatically adjudicated by the blockchain.

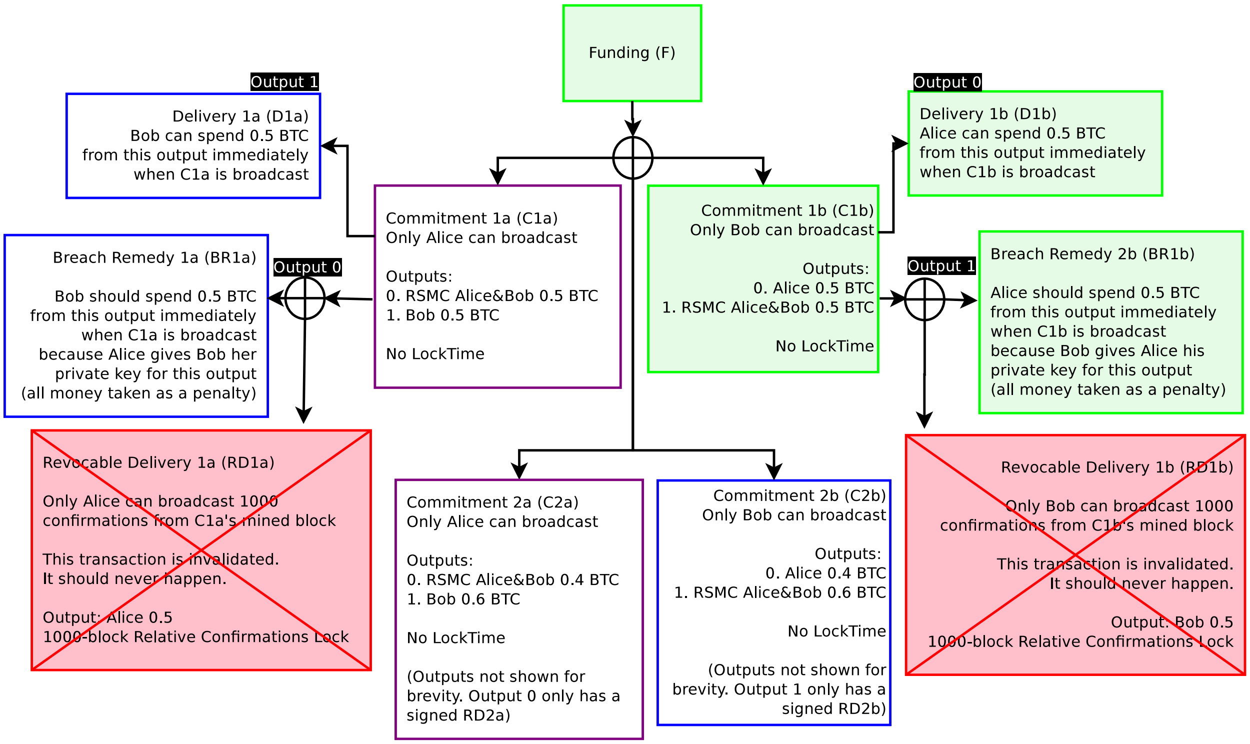

Funding (F)
Output 1
Delivery 1a (D1a)
Bob can spend 0.5 BTC
from this output immediately
when C1a is broadcast
Commitment 1a (C1a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.5 BTC
1. Bob 0.5 BTC
No LockTime
Output 0
Delivery 1b (D1b)
Alice can spend 0.5 BTC
from this output immediately
when C1b is broadcast
Commitment 1b (C1b)
Only Bob can broadcast
Outputs:
0. Alice 0.5 BTC
1. RSMC Alice&Bob 0.5 BTC
No LockTime
Output 0
Breach Remedy 1a (BR1a)
Bob should spend 0.5 BTC
from this output immediately
when C1a is broadcast
because Alice gives Bob her
private key for this output
(all money taken as a penalty)
Output 1
Breach Remedy 2b (BR1b)
Alice should spend 0.5 BTC
from this output immediately
when C1b is broadcast
because Bob gives Alice his
private key for this output
(all money taken as a penalty)
Revocable Delivery 1a (RD1a)
Only Alice can broadcast 1000
confirmations from C1a's mined block
This transaction is invalidated.
It should never happen.
Output: Alice 0.5
1000-block Relative Confirmations Lock
Commitment 2a (C2a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.4 BTC
1. Bob 0.6 BTC
No LockTime
(Outputs not shown for
brevity. Output 0 only has a
signed RD2a)
Commitment 2b (C2b)
Only Bob can broadcast
Outputs:
0. Alice 0.4 BTC
1. RSMC Alice&Bob 0.6 BTC
No LockTime
(Outputs not shown for
brevity. Output 1 only has a
signed RD2b)
Revocable Delivery 1b (RD1b)
Only Bob can broadcast 1000
confirmations from C1b's mined block
This transaction is invalidated.
It should never happen.
Output: Bob 0.5
1000-block Relative Confirmations Lock

Figure 9: Transactions in green are committed to the blockchain. Bob incorrectly broadcasts C1b (only Bob is able to broadcast C1b/C2b). Because both agreed that the current state is the C2a/C2b Commitment pair, and have attested to each party that old commitments are invalidated via Breach Remedy Transactions, Alice is able to broadcast BR1b and take all the money in the channel, provided she does it within 1000 blocks after C1b is broadcast.

However, if Alice does not broadcast BR1b within 1000 blocks, Bob may be able to steal some money, since his Revocable Delivery Transaction (RD1b) becomes valid after 1000 blocks. When an incorrect Commitment Transaction is broadcast, only the Breach Remedy Transaction can be broadcast for 1000 blocks (or whatever number of confirmations both parties agree to). After 1000 block confirmations, both the Breach Remedy (BR1b) and Revocable Delivery Transactions (RD1b) are able to be broadcast at any time. Breach Remedy transactions only have exclusivity within this predefined time period, and any time after of that is functionally an expiration of the statute of limitations —according to Bitcoin blockchain consensus, the time for dispute has ended.

For this reason, one should periodically monitor the blockchain to see if one's counterparty has broadcast an invalidated Commitment Transaction, or delegate a third party to do so. A third party can be delegated by only giving the Breach Remedy transaction to this third party. They can be incentivized to watch the blockchain broadcast such a transaction in the event of counterparty maliciousness by giving these third parties some fee in the output. Since the third party is only able to take action when the counterparty is acting maliciously, this third party does not have any power to force close of the channel.

3.3.5 Process for Creating Revocable Commitment Transactions

To create revocable Commitment Transactions, it requires proper construction of the channel from the beginning, and only signing transactions which may be broadcast at any time in the future, while ensuring that one will not lose out due to uncooperative or malicious counterparties. This requires determining which public key to use for new commitments, as using SIGHASH_NOINPUT requires using unique keys for each Commitment Transaction RSMC (and HTLC) output. We use P to designate pubkeys and K to designate the corresponding private key used to sign.

When generating the first Commitment Transaction, Alice and Bob agree to create a multisig output from a Funding Transaction with a single $multisig(P_{AliceF}, P_{BobF})$ output, funded with 0.5 BTC from Alice and Bob for a total of 1 BTC. This output is a Pay to Script Hash[16] transaction, which requires both Alice and Bob to both agree to spend from the Funding Transaction. They do not yet make the Funding Transaction (F) spendable. Additionally, P_{AliceF} and P_{BobF} are only used for the Funding Transaction, they are not used for anything else.

Since the Delivery transaction is just a P2PKH output (bitcoin addresses beginning with 1) or P2SH transaction (commonly recognized as addresses beginning with the 3) which the counterparties designate beforehand, this can be generated as an output of P_{AliceD} and P_{BobD}. For simplicity, these output addresses will remain the same throughout the channel, since its funds are fully controlled by its designated recipient after the Commitment Transaction enters the blockchain. If desired, but not necessary, both parties may update and change P_{AliceD} and P_{BobD} for future Commitment Transactions.

Both parties exchange pubkeys they intend to use for the RSMC (and HTLC described in future sections) for the Commitment Transaction. Each set of Commitment Transactions use their own public keys and are not ever reused. Both parties may already know all future pubkeys by using a BIP 0032[17] HD Wallet construction by exchanging Master Public Keys during channel construction. If they wish to generate a new Commitment Transaction pair C2a/C2b, they use multisig($P_{AliceRSMC2}$, $P_{BobRSMC2}$) for the RSMC output.

After both parties know the output values from the Commitment Transactions, both parties create the pair of Commitment Transactions, e.g. C2a/C2b, but do not exchange signatures for the Commitment Transactions. They both sign the Revocable Delivery transaction (RD2a/RD2b) and exchange the signatures. Bob signs RD1a and gives it to Alice (using $K_{BobRSMC2}$), while Alice signs RD1b and gives it to Bob (using $K_{AliceRSMC2}$).

When both parties have the Revocable Delivery transaction, they exchange signatures for the Commitment Transactions. Bob signs C1a using K_{BobF} and gives it to Alice, and Alice signs C1b using K_{AliceF} and gives it to Bob.

At this point, the prior Commitment Transaction as well as the new Commitment Transaction can be broadcast; both C1a/C1b and C2a/C2b are valid. (Note that Commitments older than the prior Commitment are invalidated via penalties.) In order to invalidate C1a and C1b, both parties exchange Breach Remedy Transaction (BR1a/BR1b) signatures for the prior commitment C1a/C1b. Alice sends BR1a to Bob using $K_{AliceRSMC1}$, and Bob sends BR1b to Alice using $K_{BobRSMC1}$. When both Breach Remedy signatures have been exchanged, the channel state is now at the current Commitment C2a/C2b and the balances are now committed.

However, instead of disclosing the BR1a/BR1b signatures, it's also possible to just disclose the private keys to the counterparty. This is more effective as described later in the key storage section. One can disclose the private keys used in one's own Commitment Transaction. For example, if Bob wishes to invalidate C1b, he sends his private keys used in C1b to Alice (he does *NOT* disclose his keys used in C1a, as that would permit coin theft). Similarly, Alice discloses all her private key outputs in C1a to Bob to invalidate C1a.

If Bob incorrectly broadcasts C1b, then because Alice has all the private keys used in the outputs of C1b, she can take the money. However, only Bob is able to broadcast C1b. To prevent this coin theft risk, Bob should destroy all old Commitment Transactions.

3.4 Cooperatively Closing Out a Channel

Both parties are able to send as many payments to their counterparty as they wish, as long as they have funds available in the channel, knowing that in the event of disagreements they can broadcast to the blockchain the current state at any time.

In the vast majority of cases, all the outputs from the Funding Transaction will never be broadcast on the blockchain. They are just there in case the other party is non-cooperative, much like how a contract is rarely enforced in the courts. A proven ability for the contract to be enforced in a deterministic manner is sufficient incentive for both parties to act honestly.

When either party wishes to close out a channel cooperatively, they will be able to do so by contacting the other party and spending from the Funding Transaction with an output of the most current Commitment Transaction directly with no script encumbering conditions. No further payments may occur in the channel.

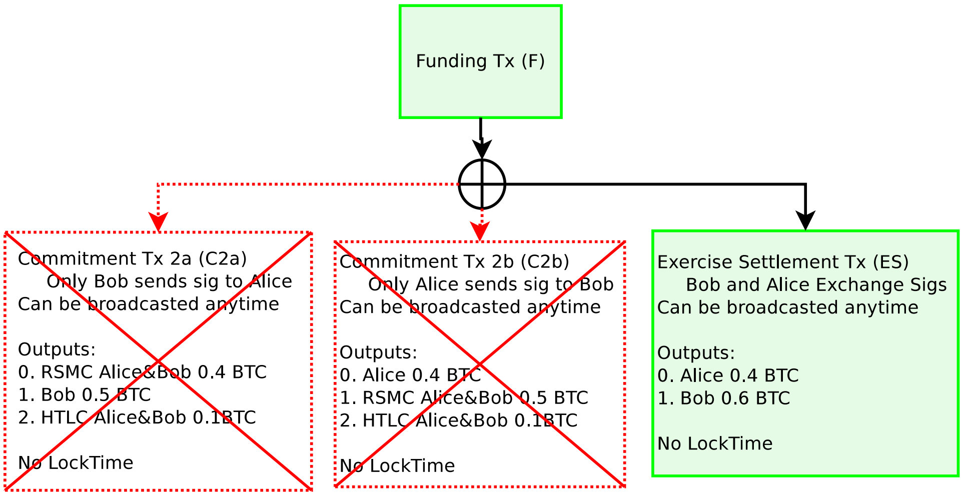

Funding Tx (F)
Commitment Tx 2a (C2a)
Only Bob sends sig to Alice
Can be broadcasted anytime
Outputs:
0. RSMC Alice&Bob 0.4 BTC
1. Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Commitment Tx 2b (C2b)
Only Alice sends sig to Bob
Can be broadcasted anytime
Outputs:
0. Alice 0.4 BTC
1. RSMC Alice&Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Exercise Settlement Tx (ES)
Bob and Alice Exchange Sigs
Can be broadcasted anytime
Outputs:
0. Alice 0.4 BTC
1. Bob 0.6 BTC
No LockTime

Figure 10: If both counterparties are cooperative, they take the balances in the current Commitment Transaction and spend from the Funding Transaction with a Exercise Settlement Transaction (ES). If the most recent Commitment Transaction gets broadcast instead, the payout (less fees) will be the same.

The purpose of closing out cooperatively is to reduce the number of transactions that occur on the blockchain and both parties will be able to receive their funds immediately (instead of one party waiting for the Revocation Delivery transaction to become valid).

Channels may remain in perpetuity until they decide to cooperatively close out the transaction, or when one party does not cooperate with another and the channel gets closed out and enforced on the blockchain.

3.5 Bidirectional Channel Implications and Summary

By ensuring channels can update only with the consent of both parties, it is possible to construct channels which perpetually exist in the blockchain. Both parties can update the balance inside the channel with whatever output balances they wish, so long as it's equal or less than the total funds committed inside the Funding Transaction; balances can move in both directions. If one party becomes malicious, either party may immediately close out the channel and broadcast the most current state to the blockchain. By using a fidelity bond construction (Revocable Delivery Transactions), if a party violates the terms of the channel, the funds will be sent to the counterparty, provided the proof of violation (Breach Remedy Transaction) is entered into the blockchain in a timely manner. If both parties are cooperative, the channel can remain open indefinitely, possibly for many years.

This type of construction is only possible because adjudication occurs programatically over the blockchain as part of the Bitcoin consensus, so one does not need to trust the other party. As a result, one's channel counterparty does not possess full custody or control of the funds.

4 Hashed Timelock Contract (HTLC)

A bidirectional payment channel only permits secure transfer of funds inside a channel. To be able to construct secure transfers using a network of channels across multiple hops to the final destination requires an additional construction, a Hashed Timelock Contract (HTLC).

The purpose of an HTLC is to allow for global state across multiple nodes via hashes. This global state is ensured by time commitments and time-based unencumbering of resources via disclosure of preimages. Transactional "locking" occurs globally via commitments, at any point in time a single participant is responsible for disclosing to the next participant whether they have knowledge of the preimage R. This construction does not require custodial trust in one's channel counterparty, nor any other participant in the network.

In order to achieve this, an HTLC must be able to create certain transactions which are only valid after a certain date, using nLockTime, as well as information disclosure to one's channel counterparty. Additionally, this data must be revocable, as one must be able to undo an HTLC.

An HTLC is also a channel contract with one's counterparty which is enforcible via the blockchain. The counterparties in a channel agree to the following terms for a Hashed Timelock Contract:

1. If Bob can produce to Alice an unknown 20-byte random input data R from a known hash H, within three days, then Alice will settle the contract by paying Bob 0.1 BTC.

2. If three days have elapsed, then the above clause is null and void and the clearing process is invalidated, both parties must not attempt to settle and claim payment after three days.

3. Either party may (and should) pay out according to the terms of this contract in any method of the participants choosing and close out this contract early so long as both participants in this contract agree.

4. Violation of the above terms will incur a maximum penalty of the funds locked up in this contract, to be paid to the non-violating counterparty as a fidelity bond.

For clarity of examples, we use days for HTLCs and block height for RSMCs. In reality, the HTLC should also be defined as a block height (e.g. 3 days is equivalent to 432 blocks).

In effect, one desires to construct a payment which is contingent upon knowledge of R by the recipient within a certain timeframe. After this timeframe, the funds are refunded back to the sender.

Similar to RSMCs, these contract terms are programatically enforced on the Bitoin blockchain and do not require trust in the counterparty to adhere to the contract terms, as all violations are penalized via unilaterally enforced fidelity bonds, which are constructed using penalty transactions spending from commitment states. If Bob knows R within three days, then he can redeem the funds by broadcasting a transaction; Alice is unable to withhold the funds in any way, because the script returns as valid when the transaction is spent on the Bitcoin blockchain.

An HTLC is an additional output in a Commitment Transaction with a unique output script:

```
OP_IF
        OP_HASH160 <Hash160(R)> OP_EQUALVERIFY
        2 <Alice2> <Bob2> OP_CHECKMULTISIG
OP_ELSE
        2 <Alice1> <Bob1> OP_CHECKMULTISIG
OP_ENDIF
```

Conceptually, this script has two possible paths spending from a single HTLC output. The first path (defined in the OP_IF) sends funds to Bob if Bob can produce R. The second path is redeemed using a 3-day timelocked refund to Alice. The 3-day timelock is enforced using nLockTime from the spending transaction.

4.1 Non-revocable HTLC Construction

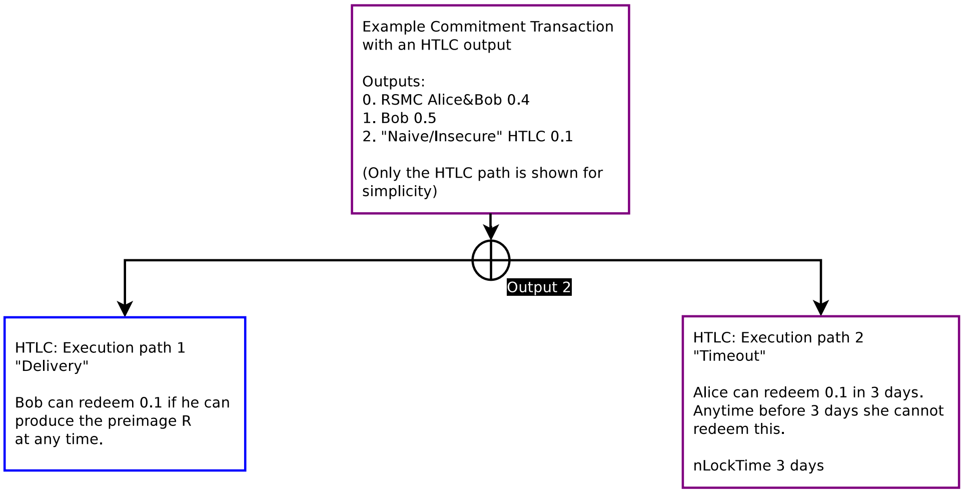

Example Commitment Transaction with an HTLC output
Outputs:
0. RSMC Alice&Bob 0.4
1. Bob 0.5
2. "Naive/Insecure" HTLC 0.1
(Only the HTLC path is shown for simplicity)
Output 2
HTLC: Execution path 1 "Delivery"
Bob can redeem 0.1 if he can produce the preimage R at any time.
HTLC: Execution path 2 "Timeout"
Alice can redeem 0.1 in 3 days. Anytime before 3 days she cannot redeem this.
nLockTime 3 days

Figure 11: This is a non-functional naive implementation of an HTLC. Only the HTLC path from the Commitment Transaction is displayed. Note that there are two possible spends from an HTLC output. If Bob can produce the preimage R within 3 days and he can redeem path 1. After three days, Alice is able to broadcast path 2. When 3 days have elapsed either is valid. This model, however, doesn't work with multiple Commitment Transactions.

If R is produced within 3 days, then Bob can redeem the funds by broadcasting the "Delivery" transaction. A requirement for the "Delivery" transaction to be valid requires R to be included with the transaction. If R is not included, then the "Delivery" transaction is invalid. However, if 3 days have elapsed, the funds can be sent back to Alice by broadcasting transaction "Timeout". When 3 days have elapsed and R has been disclosed, either transaction may be valid.

It is within both parties individual responsibility to ensure that they can get their transaction into the blockchain in order to ensure the balances are correct. For Bob, in order to receive the funds, he must either broadcast the "Delivery" transaction on the Bitcoin blockchain, or otherwise settle with Alice (while cancelling the HTLC). For Alice, she must broadcast the "Timeout" 3 days from now to receive the refund, or cancel the HTLC entirely with Bob.

Yet this kind of simplistic construction has similar problems as an incorrect bidirectional payment channel construction. When an old Commitment Transaction gets broadcast, either party may attempt to steal funds as both paths may be valid after the fact. For example, if R gets disclosed 1 year later, and an incorrect Commitment Transaction gets broadcast, both paths are valid and are redeemable by either party; the contract is not yet enforcible on the blockchain. Closing out the HTLC is absolutely necessary, because in order for Alice to get her refund, she must terminate the contract and receive her refund. Otherwise, when Bob discovers R after 3 days have elapsed, he may be able to steal the funds which should be going to Alice. With uncooperative counterparties it's not possible to terminate an HTLC without broadcasting it to the bitcoin blockchain as the uncooperative party is unwilling to create a new Commitment Transaction.

4.2 Off-chain Revocable HTLC

To be able to terminate this contract off-chain without a broadcast to the Bitcoin blockchain requires embedding RSMCs in the output, which will have a similar construction to the bidirectional channel.

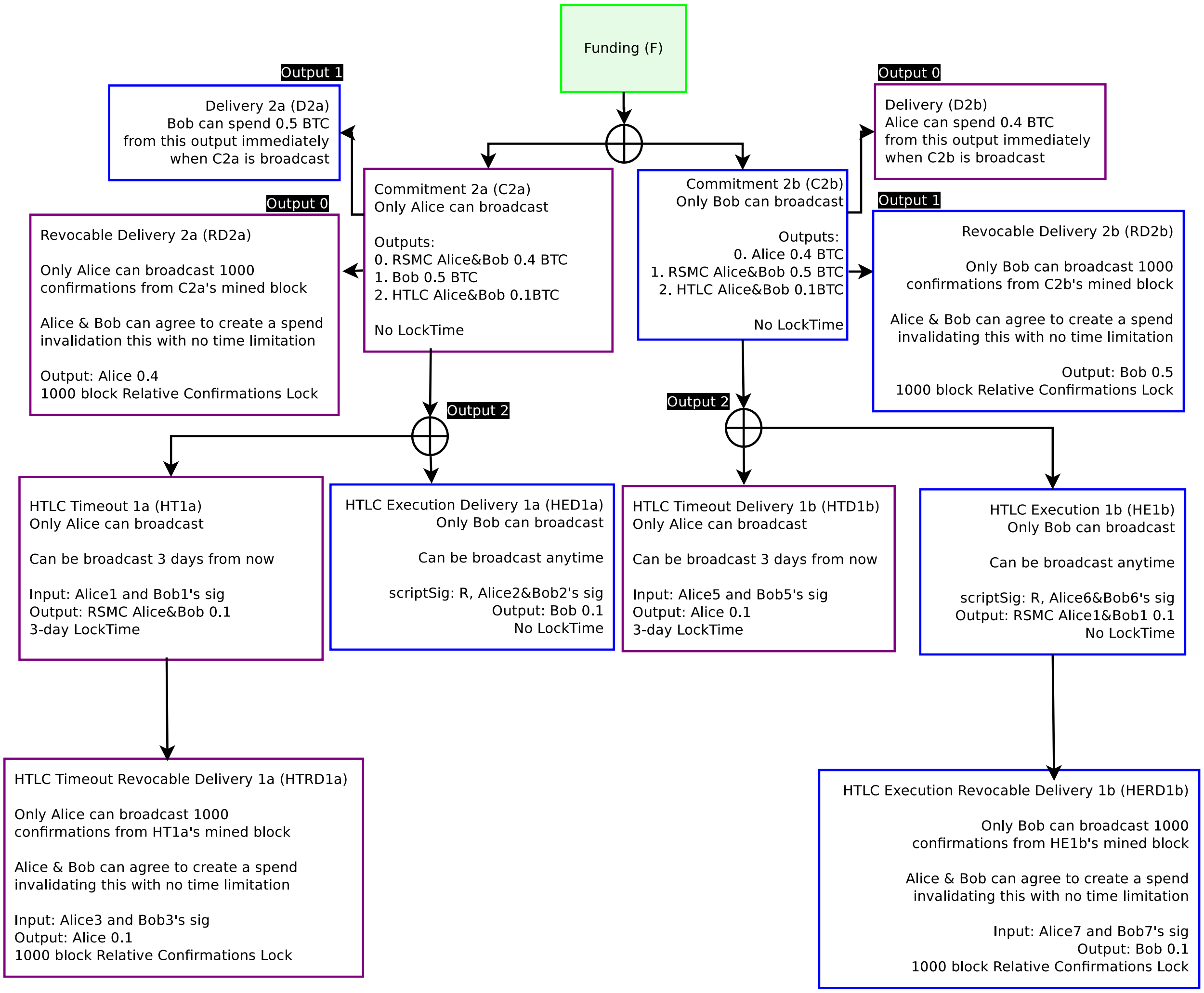

Funding (F)
Output 1
Delivery 2a (D2a)
Bob can spend 0.5 BTC
from this output immediately
when C2a is broadcast
Output 0
Delivery (D2b)
Alice can spend 0.4 BTC
from this output immediately
when C2b is broadcast
Output 0
Revocable Delivery 2a (RD2a)
Only Alice can broadcast 1000
confirmations from C2a's mined block
Alice & Bob can agree to create a spend
invalidation this with no time limitation
Output: Alice 0.4
1000 block Relative Confirmations Lock
Commitment 2a (C2a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.4 BTC
1. Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Commitment 2b (C2b)
Only Bob can broadcast
Outputs:
0. Alice 0.4 BTC
1. RSMC Alice&Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Output 1
Revocable Delivery 2b (RD2b)
Only Bob can broadcast 1000
confirmations from C2b's mined block
Alice & Bob can agree to create a spend
invalidating this with no time limitation
Output: Bob 0.5
1000 block Relative Confirmations Lock
Output 2
Output 2
HTLC Timeout 1a (HT1a)
Only Alice can broadcast
Can be broadcast 3 days from now
Input: Alice1 and Bob1's sig
Output: RSMC Alice&Bob 0.1
3-day LockTime
HTLC Execution Delivery 1a (HED1a)
Only Bob can broadcast
Can be broadcast anytime
scriptSig: R, Alice2&Bob2's sig
Output: Bob 0.1
No LockTime
HTLC Timeout Delivery 1b (HTD1b)
Only Alice can broadcast
Can be broadcast 3 days from now
Input: Alice5 and Bob5's sig
Output: Alice 0.1
3-day LockTime
HTLC Execution 1b (HE1b)
Only Bob can broadcast
Can be broadcast anytime
scriptSig: R, Alice6&Bob6's sig
Output: RSMC Alice1&Bob1 0.1
No LockTime
HTLC Timeout Revocable Delivery 1a (HTRD1a)
Only Alice can broadcast 1000
confirmations from HT1a's mined block
Alice & Bob can agree to create a spend
invalidating this with no time limitation
Input: Alice3 and Bob3's sig
Output: Alice 0.1
1000 block Relative Confirmations Lock
HTLC Execution Revocable Delivery 1b (HERD1b)
Only Bob can broadcast 1000
confirmations from HE1b's mined block
Alice & Bob can agree to create a spend
invalidating this with no time limitation
Input: Alice7 and Bob7's sig
Output: Bob 0.1
1000 block Relative Confirmations Lock

Figure 12: If Alice broadcasts C2a, then the left half will execute. If Bob broadcasts C2b, then the right half will execute. Either party may broadcast their Commitment transaction at any time. HTLC Timeout is only valid after 3 days. HTLC Executions can only be broadcast if the preimage to the hash R is known. Prior Commitments (and their dependent transactions) are not displayed for brevity.

Presume Alice and Bob wish to update their balance in the channel at Commitment 1 with a balance of 0.5 to Alice and 0.5 to Bob.

Alice wishes to send 0.1 to Bob contingent upon knowledge of R within 3 days, after 3 days she wants her money back if Bob does not produce R.

The new Commitment Transaction will have a full refund of the current balance to Alice and Bob (Outputs 0 and 1), with output 2 being the HTLC, which describes the funds in transit. As 0.1 will be encumbered in an HTLC, Alice's balance is reduced to 0.4 and Bob's remains the same at

0.5.

This new Commitment Transaction (C2a/C2b) will have an HTLC output with two possible spends. Each spend is different depending on each counterparty's version of the Commitment Transaction. Similar to the bidirectional payment channel, when one party broadcasts their Commitment, payments to the counterparty will be assumed to be valid and not invalidated. This can occur because when one broadcasts a Commitment Transaction, one is attesting this is the most recent Commitment Transaction. If it is the most recent, then one is also attesting that the HTLC exists and was not invalidated before, so potential payments to one's counterparty should be valid.

Note that HTLC transaction names (beginning with the letter H) will begin with the number 1, whose values do not correlate with Commitment Transactions. This is simply the first HTLC transaction. HTLC transactions may persist between Commitment Transactions. Each HTLC has 4 keys per side of the transaction (C2a and C2b) for a total of 8 keys per counterparty.

The HTLC output in the Commitment Transaction has two sets of keys per counterparty in the output.

For Alice's Commitment Transaction (C2a), the HTLC output script requires $multisig(P_{Alice2}, P_{Bob2})$ encumbered by disclosure of R, as well as $multisig(P_{Alice1}, P_{Bob1})$ with no encumbering.

For Bob's Commitment Transaction (C2b), the HTLC output script requires $multisig(P_{Alice6}, P_{Bob6})$ encumbered by disclosure of R, as well as $multisig(P_{Alice5}, P_{Bob5})$ with no encumbering.

The HTLC output states are different depending upon which Commitment Transaction is broadcast.

4.2.1 HTLC when the Sender Broadcasts the Commitment Transaction

For the sender (Alice), the "Delivery" transaction is sent as an HTLC Execution Delivery transaction (HED1a), which is not encumbered in an RSMC. It assumes that this HTLC has never been terminated off-chain, as Alice is attesting that the broadcasted Commitment Transaction is the most recent. If Bob can produce the preimage R, he will be able to redeem funds from the HTLC after the Commitment Transaction is broadcast on the blockchain.

This transaction consumes $multisig(P_{Alice2}, P_{Bob2})$ if Alice broadcasts her Commitment C2a. Only Bob can broadcast HED1a since only Alice gave her signature for HED1a to Bob.

However, if 3 days have elapsed since forming the HTLC, then Alice will be able broadcast a "Timeout" transaction, the HTLC Timeout transaction (HT1a). This transaction is an RSMC. It consumes the output $multisig(P_{Alice1}, P_{Bob1})$ without requiring disclosure of R if Alice broadcasts C2a. This transaction cannot enter into the blockchain until 3 days have elapsed. The output for this transaction is an RSMC with $multisig(P_{Alice3}, P_{Bob3})$ with relative maturity of 1000 blocks, and $multisig(P_{Alice4}, P_{Bob4})$ with no requirement for confirmation maturity. Only Alice can broadcast HT1a since only Bob gave his signature for HT1a to Alice.

After HT1a enters into the blockchain and 1000 block confirmations occur, an HTLC Timeout Revocable Delivery transaction (HTRD1a) may be broadcast by Alice which consumes $multisig(P_{Alice3}, P_{Bob3})$. Only Alice can broadcast HTRD1a 1000 blocks after HT1a is broadcast since only Bob gave his signature for HTRD1a to Alice. This transaction can be revocable when another transaction supersedes HTRD1a using $multisig(P_{Alice4}, P_{Bob4})$ which does not have any block maturity requirements.

4.2.2 HTLC when the Receiver Broadcasts the Commitment Transaction

For the potential receiver (Bob), the "Timeout" of receipt is refunded as an HTLC Timeout Delivery transaction (HTD1b). This transaction directly refunds the funds to the original sender (Alice) and is not encumbered in an RSMC. It assumes that this HTLC has never been terminated off-chain, as Bob is attesting that the broadcasted Commitment Transaction (C2b) is the most recent. If 3 days have elapsed, Alice can broadcast HTD1b and take the refund. This transaction consumes $multisig(P_{Alice5}, P_{Alice5})$ if Bob broadcasts C2b. Only Alice can broadcast HTD1b since Bob gave his signature for HTD1b to Alice.

However, if HTD1b is not broadcast (3 days have not elapsed) and Bob knows the preimage R, then Bob will be able to broadcast the HTLC Execution transaction (HE1b) if he can produce R. This transaction is an

RSMC. It consumes the output $multisig(P_{Alice6}, P_{Bob6})$ and requires disclosure of R if Bob broadcasts C2b. The output for this transaction is an RSMC with $multisig(P_{Alice7}, P_{Bob7})$ with relative maturity of 1000 blocks, and $multisig(P_{Alice8}, P_{Bob8})$ which does not have any block maturity requirements. Only Bob can broadcast HE1b since only Alice gave her signature for HE1b to Bob.

After HE1b enters into the blockchain and 1000 block confirmations occur, an HTLC Execution Revocable Delivery transaction (HERD1b) may be broadcast by Bob which consumes $multisig(P_{Alice7}, P_{Bob7})$. Only Bob can broadcast HERD1b 1000 blocks after HE1b is broadcast since only Alice gave her signature for HERD1b to Bob. This transaction can be revocable when another transaction supersedes HERD1b using $multisig(P_{Alice8}, P_{Bob8})$ which does not have any block maturity requirements.

4.3 HTLC Off-chain Termination

After an HTLC is constructed, to terminate an HTLC off-chain requires both parties to agree on the state of the channel. If the recipient can prove knowledge of R to the counterparty, the recipient is proving that they are able to immediately close out the channel on the Bitcoin blockchain and receive the funds. At this point, if both parties wish to keep the channel open, they should terminate the HTLC off-chain and create a new Commitment Transaction reflecting the new balance.

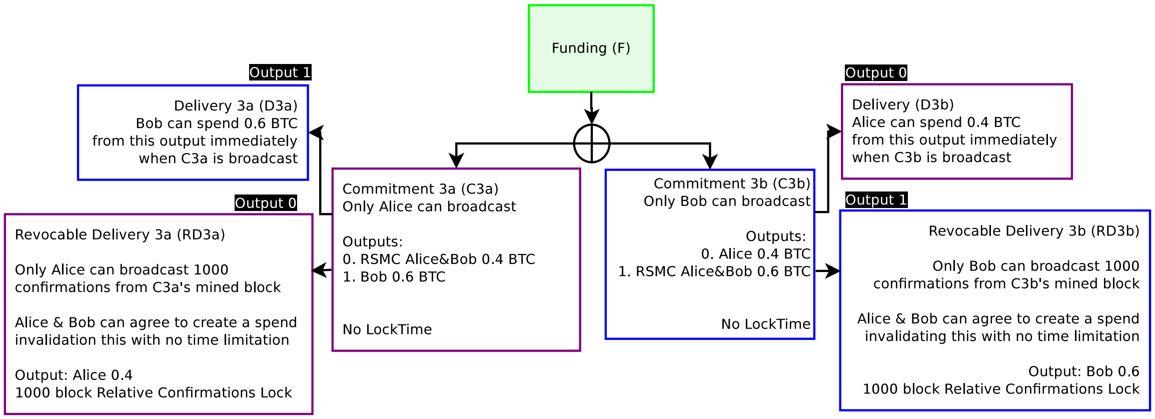

Funding (F)
Output 1
Delivery 3a (D3a)
Bob can spend 0.6 BTC
from this output immediately
when C3a is broadcast
Output 0
Revocable Delivery 3a (RD3a)
Only Alice can broadcast 1000
confirmations from C3a's mined block
Alice & Bob can agree to create a spend
invalidation this with no time limitation
Output: Alice 0.4
1000 block Relative Confirmations Lock
Commitment 3a (C3a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.4 BTC
1. Bob 0.6 BTC
No LockTime
Commitment 3b (C3b)
Only Bob can broadcast
Outputs:
0. Alice 0.4 BTC
1. RSMC Alice&Bob 0.6 BTC
No LockTime
Output 0
Delivery (D3b)
Alice can spend 0.4 BTC
from this output immediately
when C3b is broadcast
Output 1
Revocable Delivery 3b (RD3b)
Only Bob can broadcast 1000
confirmations from C3b's mined block
Alice & Bob can agree to create a spend
invalidating this with no time limitation
Output: Bob 0.6
1000 block Relative Confirmations Lock

Figure 13: Since Bob proved to Alice he knows R by telling Alice R, Alice is willing to update the balance with a new Commitment Transaction. The payout will be the same whether C2 or C3 is broadcast at this time.

Similarly, if the recipient is not able to prove knowledge of R by disclosing R, both parties should agree to terminate the HTLC and create a new Commitment Transaction with the balance in the HTLC refunded to the sender.

If the counterparties cannot come to an agreement or become otherwise unresponsive, they should close out the channel by broadcasting the necessary channel transactions on the Bitcoin blockchain.

However, if they are cooperative, they can do so by first generating a new Commitment Transaction with the new balances, then invalidate the prior Commitment by exchanging Breach Remedy transactions (BR2a/BR2b). Additionally, if they are terminating a particular HTLC, they should also exchange some of their own private keys used in the HTLC transactions.

For example, Alice wishes to terminate the HTLC, Alice will disclose K_{Alice1} and K_{Alice4} to Bob. Correspondingly if Bob wishes to terminate the HTLC, Bob will disclose K_{Bob6} and K_{Bob8} to Alice. After the private keys are disclosed to the counterparty, if Alice broadcasts C2a, Bob will be able to take all the funds from the HTLC immediately. If Bob broadcasts C2b, Alice will be able to take all funds from the HTLC immediately. Note that when an HTLC is terminated, the older Commitment Transaction must be revoked as well.

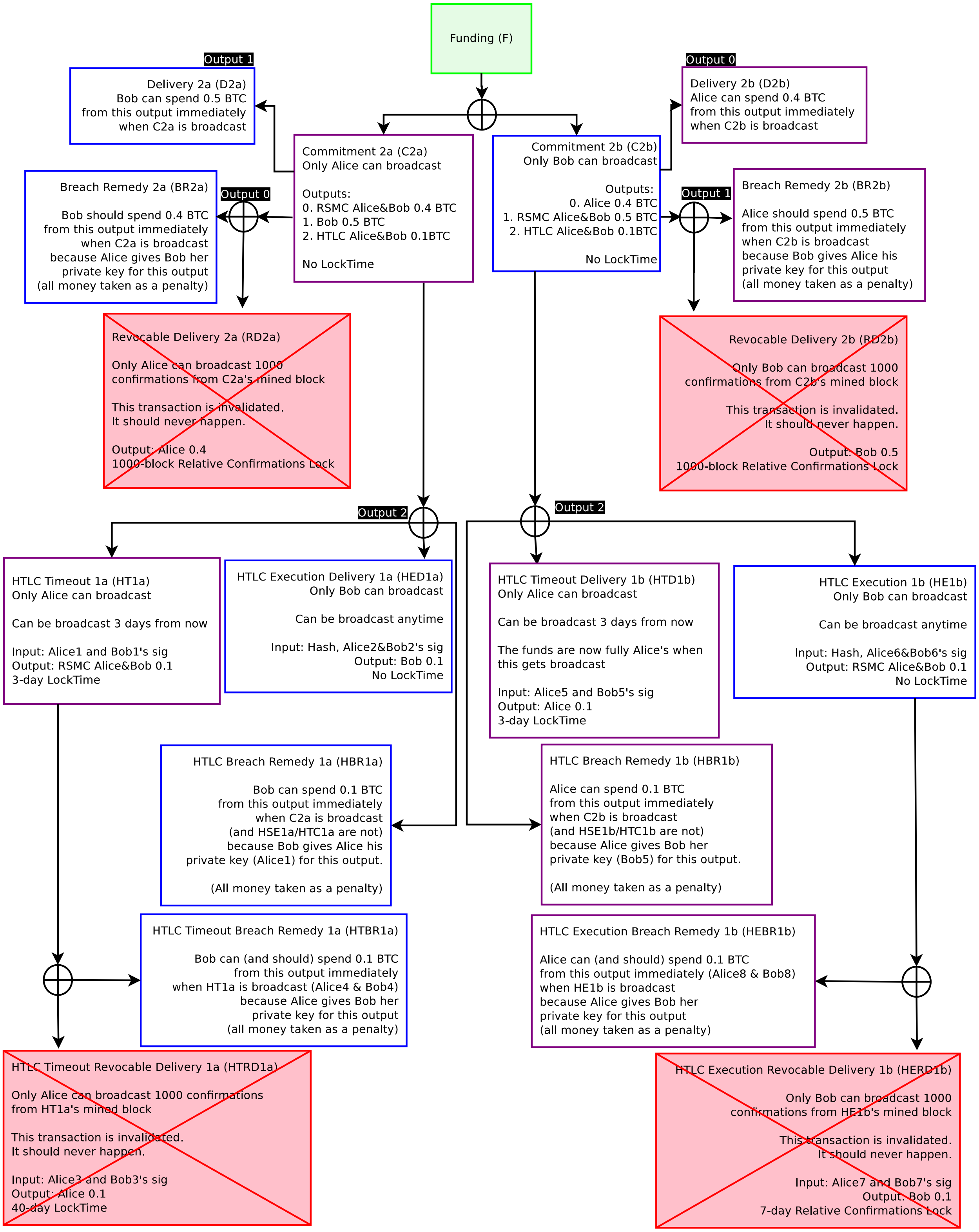

Funding (F)
Output 1
Delivery 2a (D2a)
Bob can spend 0.5 BTC
from this output immediately
when C2a is broadcast
Output 0
Delivery 2b (D2b)
Alice can spend 0.4 BTC
from this output immediately
when C2b is broadcast
Commitment 2a (C2a)
Only Alice can broadcast
Outputs:
0. RSMC Alice&Bob 0.4 BTC
1. Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Commitment 2b (C2b)
Only Bob can broadcast
Outputs:
0. Alice 0.4 BTC
1. RSMC Alice&Bob 0.5 BTC
2. HTLC Alice&Bob 0.1BTC
No LockTime
Breach Remedy 2a (BR2a)
Bob should spend 0.4 BTC
from this output immediately
when C2a is broadcast
because Alice gives Bob her
private key for this output
(all money taken as a penalty)
Output 0
Output 1
Breach Remedy 2b (BR2b)
Alice should spend 0.5 BTC
from this output immediately
when C2b is broadcast
because Bob gives Alice his
private key for this output
(all money taken as a penalty)
Revocable Delivery 2a (RD2a)
Only Alice can broadcast 1000
confirmations from C2a's mined block
This transaction is invalidated.
It should never happen.
Output: Alice 0.4
1000-block Relative Confirmations Lock
Revocable Delivery 2b (RD2b)
Only Bob can broadcast 1000
confirmations from C2b's mined block
This transaction is invalidated.
It should never happen.
Output: Bob 0.5
1000-block Relative Confirmations Lock
Output 2
Output 2
HTLC Timeout 1a (HT1a)
Only Alice can broadcast
Can be broadcast 3 days from now
Input: Alice1 and Bob1's sig
Output: RSMC Alice&Bob 0.1
3-day LockTime
HTLC Execution Delivery 1a (HED1a)
Only Bob can broadcast
Can be broadcast anytime
Input: Hash, Alice2&Bob2's sig
Output: Bob 0.1
No LockTime
HTLC Timeout Delivery 1b (HTD1b)
Only Alice can broadcast
Can be broadcast 3 days from now
The funds are now fully Alice's when
this gets broadcast
Input: Alice5 and Bob5's sig
Output: Alice 0.1
3-day LockTime
HTLC Execution 1b (HE1b)
Only Bob can broadcast
Can be broadcast anytime
Input: Hash, Alice6&Bob6's sig
Output: RSMC Alice&Bob 0.1
No LockTime
HTLC Breach Remedy 1a (HBR1a)
Bob can spend 0.1 BTC
from this output immediately
when C2a is broadcast
(and HSE1a/HTC1a are not)
because Bob gives Alice his
private key (Alice1) for this output.
(All money taken as a penalty)
HTLC Breach Remedy 1b (HBR1b)
Alice can spend 0.1 BTC
from this output immediately
when C2b is broadcast
(and HSE1b/HTC1b are not)
because Alice gives Bob her
private key (Bob5) for this output.
(All money taken as a penalty)
HTLC Timeout Breach Remedy 1a (HTBR1a)
Bob can (and should) spend 0.1 BTC
from this output immediately
when HT1a is broadcast (Alice4 & Bob4)
because Alice gives Bob her
private key for this output
(all money taken as a penalty)
HTLC Execution Breach Remedy 1b (HEBR1b)
Alice can (and should) spend 0.1 BTC
from this output immediately (Alice8 & Bob8)
when HE1b is broadcast
because Alice gives Bob her
private key for this output
(all money taken as a penalty)
HTLC Timeout Revocable Delivery 1a (HTRD1a)
Only Alice can broadcast 1000 confirmations
from HT1a's mined block
This transaction is invalidated.
It should never happen.
Input: Alice3 and Bob3's sig
Output: Alice 0.1
40-day LockTime
HTLC Execution Revocable Delivery 1b (HERD1b)
Only Bob can broadcast 1000
confirmations from HE1b's mined block
This transaction is invalidated.
It should never happen.
Input: Alice7 and Bob7's sig
Output: Bob 0.1
7-day Relative Confirmations Lock

Figure 14: A fully revoked Commitment Transaction and terminated HTLC. If either party broadcasts Commitment 2, they will lose all their money to the counterparty. Other commitments (e.g. if Commitment 3 is the current Commitment) are not displayed for brevity.

Since both parties are able to prove the current state to each other, they can come to agreement on the current balance inside the channel. Since they may broadcast the current state on the blockchain, they are able to come to agreement on netting out and terminating the HTLC with a new Commitment Transaction.

4.4 HTLC Formation and Closing Order

To create a new HTLC, it is the same process as creating a new Commitment Transaction, except the signatures for the HTLC are exchanged before the new Commitment Transaction's signatures.

To close out an HTLC, the process is as follows (from C2 to C3):

1. Alice signs and sends her signature for RD3b and C3b. At this point Bob can elect to broadcast C3b or C2b (with the HTLC) with the same payout. Bob is willing after receiving C3b to close out C2b.

2. Bob signs and sends his signature for RD3a and C3a, as well as his private keys used for Commitment 2 and the HTLC being terminated; he sends Alice $K_{BobRSMC2}$, K_{Bob5}, and K_{Bob8}. At this point Bob should only broadcast C3b and should not broadcast C2b as he will lose all his money if he does so. Bob has fully revoked C2b and the HTLC. Alice is willing after receiving C3a to close out C2b.

3. Alice signs and sends her signature for RD3b and C3b, as well as her private keys used for Commitment 2 and the HTLC being terminated; she sends Bob $K_{AliceRSMC2}$, K_{Bob1}, and K_{Bob4}. At this point neither party should broadcast Commitment 2, if they do so, their funds will be going to the counterparty. The old Commitment and old HTLC are now revoked and fully terminated. Only the new Commitment 3 remains, which does not have an HTLC.

When the HTLC has been closed, the funds are updated so that the present balance in the channel is what would occur had the HTLC contract been completed and broadcast on the blockchain. Instead, both parties elect to do off-chain novation and update their payments inside the channel.

It is absolutely necessary for both parties to complete off-chain novation within their designated time window. For the receiver (Bob), he must know R and update his balance with Alice within 3 days (or whatever time was selected), else Alice will be able to redeem it within 3 days. For Alice, very soon after her timeout becomes valid, she must novate or broadcast the HTLC Timeout transaction. She must also novate or broadcast the HTLC Timeout Revocable Delivery transaction as soon as it becomes valid. If the counterparty is unwilling to novate or is stalling, then one must broadcast the current channel state, including HTLC transactions) onto the Bitcoin blockchain.

The amount of time flexibility with these offers to novate are dependent upon one's contingent dependencies on the hashlock R. If one establishes a contract that the HTLC must be resolved within 1 day, then if the transaction times out Alice must resolve it by day 4 (3 days plus 1), else Alice risks losing funds.

5 Key Storage

Keys are generated using BIP 0032 Hierarchical Deterministic Wallets[17]. Keys are pre-generated by both parties. Keys are generated in a merkle tree and are very deep within the tree. For instance, Alice pre-generates one million keys, each key being a child of the previous key. Alice allocates which keys to use according to some deterministic manner. For example, she starts with the child deepest in the tree to generate many sub-keys for day 1. This key is used as a master key for all keys generated on day 1. She gives Bob the address she wishes to use for the next transaction, and discloses the private key to Bob when it becomes invalidated. When Alice discloses to Bob all private keys derived from the day 1 master key and does not wish to continue using that master key, she can disclose the day 1 master key to Bob. At this point, Bob does not need to store all the keys derived from the day 1 master key. Bob does the same for Alice and gives her his day 1 key.

When all Day 2 private keys have been exchanged, for example by day 5, Alice discloses her Day 2 key. Bob is able to generate the Day 1 key from the Day 2 key, as the Day 1 key is a child of the Day 2 key as well.

If a counterparty broadcasts the wrong Commitment Transaction, which private key to use in a transaction to recover funds can either be brute forced, or if both parties agree, they can use the sequence id number when creating the transaction to identify which sets of keys are used.

This enables participants in a channel to have prior output states (transactions) invalidated by both parties without using much data at all. By disclosing private keys pre-arranged in a merkle-tree, it is possible to invalidate millions of old transactions with only a few kilobytes of data per channel. Core channels in the Lightning Network can conduct billions of transactions without a need for significant storage costs.

6 Blockchain Transaction Fees for Bidirectional Channels

It is possible for each participant to generate different versions of transactions to ascribe blame as to who broadcast the transaction on the blockchain. By having knowledge of who broadcast a transaction and the ability to ascribe blame, a third party service can be used to hold fees in a 2-of-3 multisig escrow. If one wishes to broadcast the transaction chain instead of agreeing to do a Funding Close or replacement with a new Commitment Transaction, one would communicate with the third party and broadcast the chain to the blockchain. If the counterparty refuses the notice from the third party to cooperate, the penalty is rewarded to the non-cooperative party. In most instances, participants may be indifferent to the transaction fees in the event of an uncooperative counterparty.

One should pick counterparties in the channel who will be cooperative, but is not an absolute necessity for the system to function. Note that this does not require trust among the rest of the network, and is only relevant for the comparatively minor transaction fees. The less trusted party may just be the one responsible for transaction fees.

The Lightning Network fees will likely be significantly lower than blockchain transaction fees. The fees are largely derived from the time-value of locking up funds for a particular route, as well as paying for the chance of channel close on the blockchain. These should be significantly lower than on-chain transactions, as many transactions on a Lightning Network channel can be settled into one single blockchain transaction. With a sufficiently robust and interconnected network, the fees should asymptotically approach negligibility for many types of transactions. With cheap fees and fast transactions, it will be possible to build scalable micropayments, even amongst high-frequency systems such as Internet of Things applications or per-unit micro-billing.

7 Pay to Contract

It is possible construct a cryptographically provable "Delivery Versus Payment" contract, or pay-to-contract[18], as proof of payment. This proof can be established as knowledge of the input R from hash(R) as payment of a certain value. By embedding a clause into the contract between the buyer and seller stating that knowing R is proof of funds sent, the recipient of funds has no incentive to disclose R unless they have certainty that they will receive payment. When the funds eventually get pulled from the buyer by their counterparty in their micropayment channel, R is disclosed as part of that pull of funds. One can design paper legal documents that specify that knowledge or disclosure of R implies fulfillment of payment. The sender can then arrange a cryptographically signed contract with knowledge of inputs for hashes treated as fulfillment of the paper contract before payment occurs.

8 The Bitcoin Lightning Network

By having a micropayment channel with contracts encumbered by hashlocks and timelocks, it is possible to clear transactions over a multi-hop payment network using a series of decrementing timelocks without additional central clearinghouses.

Traditionally, financial markets clear transactions by transferring the obligation for delivery at a central point and settle by transferring ownership through this central hub. Bank wire and fund transfer systems (such as ACH and the Visa card network), or equities clearinghouses (such as the DTCC) operate in this manner.

As Bitcoin enables programmatic money, it is possible to create transactions without contacting a central clearinghouse. Transactions can execute off-chain with no third party which collects all funds before disbursing it – only transactions with uncooperative channel counterparties become automatically adjudicated on the blockchain.

The obligation to deliver funds to an end-recipient is achieved through a process of chained delegation. Each participant along the path assumes the obligation to deliver to a particular recipient. Each participant passes on this obligation to the next participant in the path. The obligation of each subsequent participant along the path, defined in their respective HTLCs, has a shorter time to completion compared to the prior participant. This way each participant is sure that they will be able to claim funds when the obligation is sent along the path.

Bitcoin Transaction Scripting, a form of what some call an implementation of "Smart Contracts"[19], enables systems without trusted custodial clearinghouses or escrow services.

8.1 Decrementing Timelocks

Presume Alice wishes to send 0.001 BTC to Dave. She locates a route through Bob and Carol. The transfer path would be Alice to Bob to Carol to Dave.

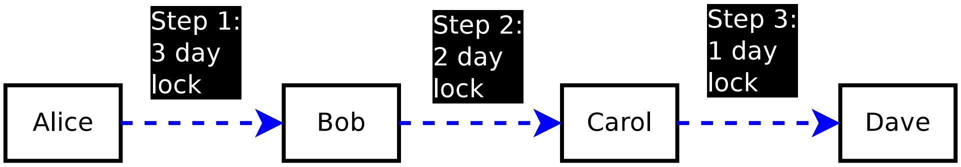


Figure 15: Payment over the Lightning Network using HTLCs.

When Alice sends payment to Dave through Bob and Carol, she requests from Dave hash(R) to use for this payment. Alice then counts the amount of hops until the recipient and uses that as the HTLC expiry. In this case, she sets the HTLC expiry at 3 days. Bob then creates an HTLC with Carol with an expiry of 2 days, and Carol does the same with Dave with an expiry of 1 day. Dave is now free to disclose R to Carol, and both parties will likely agree to immediate settlement via novation with a replacement Commitment Transaction. This then occurs step-by-step back to Alice. Note that this occurs off-chain, and nothing is broadcast to the blockchain when all parties are cooperative.

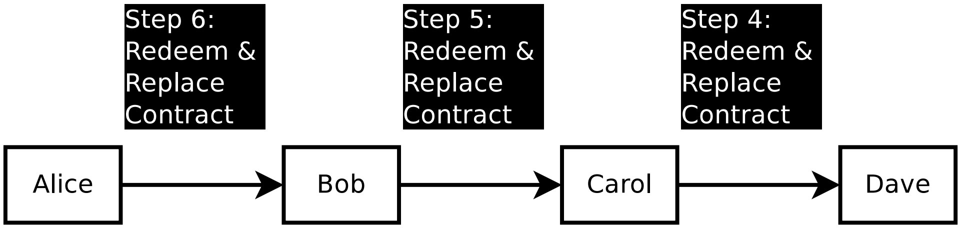


Figure 16: Settlement of HTLC, Alice's funds get sent to Dave.

Decrementing timelocks are used so that all parties along the path know that the disclosure of R will allow the disclosing party to pull funds, since they will at worst be pulling funds after the date whereby they must receive R. If Dave does not produce R within 1 day to Carol, then Carol will be able to close out the HTLC. If Dave broadcasts R after 1 day, then he will not be able to pull funds from Carol. Carol's responsibility to Bob occurs on day 2, so Carol will never be responsible for payment to Dave without an ability to pull funds from Bob provided that she updates her transaction with Dave via transmission to the blockchain or via novation.

In the event that R gets disclosed to the participants halfway through expiry along the path (e.g. day 2), then it is possible for some parties along the path to be enriched. The sender will be able to know R, so due to Pay to Contract, the payment will have been fulfilled even though the receiver did not receive the funds. Therefore, the receiver must never disclose R unless they have received an HTLC from their channel counterparty; they are guaranteed to receive payment from one of their channel counterparties upon disclosure of the preimage.

In the event a party outright disconnects, the counterparty will be responsible for broadcasting the current Commitment Transaction state in the channel to the blockchain. Only the failed non-responsive channel state gets closed out on the blockchain, all other channels should continue to update their Commitment Transactions via novation inside the channel. Therefore, counterparty risk for transaction fees are only exposed to direct channel counterparties. If a node along the path decides to become unresponsive, the participants not directly connected to that node suffer only decreased time-value of their funds by not conducting early settlement before the HTLC close.

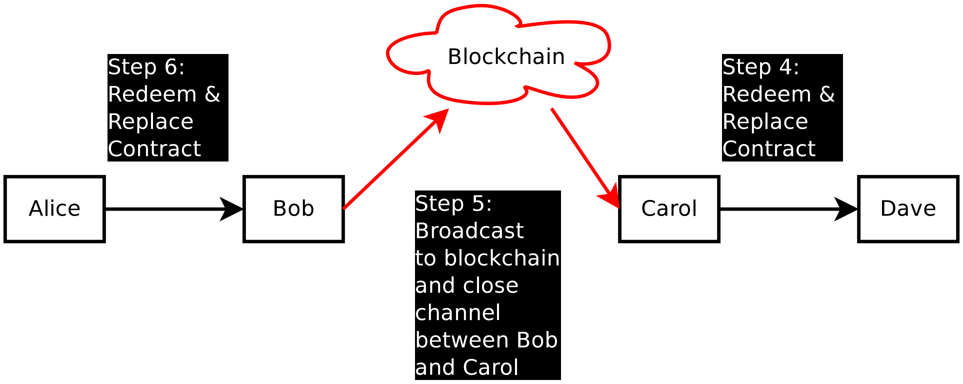

Blockchain
Step 6: Redeem & Replace Contract
Step 4: Redeem & Replace Contract
Alice
Bob
Carol
Dave
Step 5: Broadcast to blockchain and close channel between Bob and Carol

Figure 17: Only the non-responsive channels get broadcast on the blockchain, all others are settled off-chain via novation.

8.2 Payment Amount

It is preferable to use a small payment per HTLC. One should not use an extremely high payment, in case the payment does not fully route to its destination. If the payment does not reach its destination and one of the participants along the path is uncooperative, it is possible that the sender must wait until the expiry before receiving a refund. Delivery may be lossy, similar to packets on the internet, but the network cannot outright steal funds in transit. Since transactions don't hit the blockchain with cooperative channel counterparties, it is recommended to use as small of a payment as possible. A tradeoff exists between locking up transaction fees on each hop versus the desire to use as small a transaction amount as possible (the latter of which may incur higher total fees). Smaller transfers with more intermediaries imply a higher percentage paid as Lightning Network fees to the intermediaries.

8.3 Clearing Failure and Rerouting

If a transaction fails to reach its final destination, the receiver should send an equal payment to the sender with the same hash, but not disclose R. This will net out the disclosure of the hash for the sender, but may not for the receiver. The receiver, who generated the hash, should discard R and never broadcast it. If one channel along the path cannot be contacted, then the channels may elect to wait until the path expires, which all participants will likely close out the HTLC as unsettled without any payment with a new Commitment Transaction.

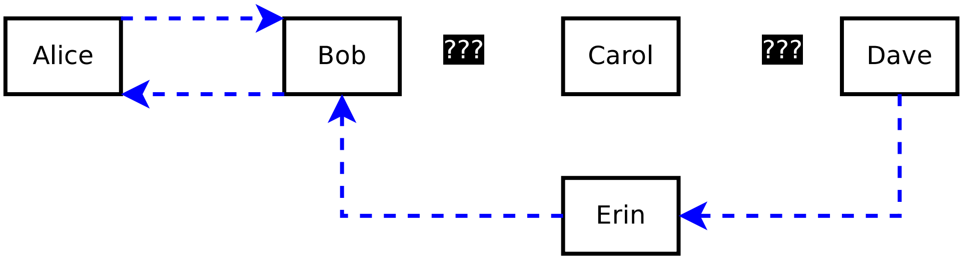


Figure 18: Dave creates a path back to Alice after Alice fails to send funds to Dave, because Carol is uncooperative. The input R from hash(R) is never brodcast by Dave, because Carol did not complete her actions. If R was broadcast, Alice will break-even. Dave, who controls R should never broadcast R because he may not receive funds from Carol, he should let the contracts expire. Alice and Bob have the option to net out and close the contract early, as well, in this diagram.

If the refund route is the same as the payment route, and there are no half-signed contracts whereby one party may be able to steal funds, it is possible to outright cancel the transaction by replacing it with a new Commitment Transaction starting with the most recent node who participated in the HTLC.

It is also possible to clear out a channel by creating an alternate route path in which payment will occur in the opposite direction (netting out to zero) and/or creating an entirely alternate route for the payment path. This will create a time-value of money for disclosing inputs to hashes on the Lightning Network. Participants may specialize in high connectivity between nodes and offering to offload contract hashlocks from other nodes for a fee. These participants will agree to payments which net out to zero (plus fees), but are loaning bitcoins for a set time period. Most likely, these entities with low demand for channel resources will be end-users who are already connected to multiple well-connected nodes. When an end-user connects to a node, the node may ask the client to lock up their funds for several days to another channel the client has established for a fee. This can be achieved by having the new transactions require a new hash(Y) from input Y in addition to the existing hash which may be generated by any participant, but must disclose Y only after a full circle is established. The new participant has the same responsibility as well as the same timelocks as the old participant being replaced. It is also possible that the one new participant replaces multiple hops.

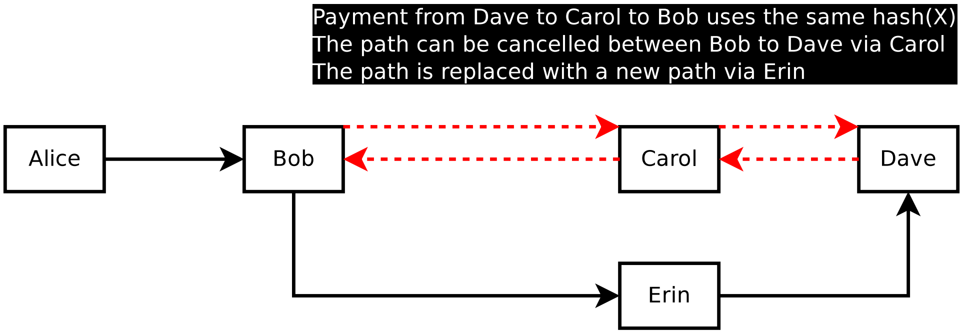

Payment from Dave to Carol to Bob uses the same hash(X)
The path can be cancelled between Bob to Dave via Carol
The path is replaced with a new path via Erin
Alice
Bob
Carol
Dave
Erin

Figure 19: Erin is connected to both Bob and Dave. If Bob wishes to free up his channel with Carol, since that channel is active and very profitable, Bob can offload the payment to Dave via Erin. Since Erin has extra bitcoin available, she will be able to collect some fee for offloading the channel between Bob and Carol as well as between Carol and Dave. The channels between Bob and Carol as well as Carol and Dave are undone and no longer have the HTLC, nor has payment occurred on that path. Payment will occur on the path involving Erin. This is achieved by creating a new payment from Dave to Carol to Bob contingent upon Erin constructing an HTLC. The payment in dashed lines (red) are netted out to zero and settled via a new Commitment Contract.

8.4 Payment Routing

It is theoretically possible to build a route map implicitly from observing 2-of-2 multisigs on the blockchain to build a routing table. Note, however, this is not feasible with pay-to-script-hash transaction outputs, which can be resolved out-of-band from the bitcoin protocol via a third party routing service. Building a routing table will become necessary for large operators (e.g. BGP, Cjdns). Eventually, with optimizations, the network will look a lot like the correspondent banking network, or Tier-1 ISPs. Similar to how packets still reach their destination on your home network connection, not all participants need to have a full routing table. The core Tier-1 routes can be online all the time —while nodes at the edges, such as average users, would be connected intermittently.

Node discovery can occur along the edges by pre-selecting and offering partial routes to well-known nodes.

8.5 Fees

Lightning Network fees, which differ from blockchain fees, are paid directly between participants within the channel. The fees pay for the time-value of money for consuming the channel for a determined maximum period of time, and for counterparty risk of non-communication.

Counterparty risk for fees only exist with one's direct channel counterparty. If a node two hops away decides to disconnect and their transaction gets broadcast on the blockchain, one's direct counterparties should not broadcast on the blockchain, but continue to update via novation with a new Commitment Transaction. See the Decrementing Timelocks entry in the HTLC section for more information about counterparty risk.

The time-value of fees pays for consuming time (e.g. 3 days) and is conceptually equivalent to a gold lease rate without custodial risk; it is the time-value for using up the access to money for a very short duration. Since certain paths may become very profitable in one direction, it is possible for fees to be negative to encourage the channel to be available for those profitable paths.

9 Risks

The primary risks relate to timelock expiration. Additionally, for core nodes and possibly some merchants to be able to route funds, the keys must be held online for lower latency. However, end-users and nodes are able to keep their private keys firewalled off in cold storage.

9.1 Improper Timelocks

Participants must choose timelocks with sufficient amounts of time. If insufficient time is given, it is possible that timelocked transactions believed to be invalid will become valid, enabling coin theft by the counterparty. There is a trade-off between longer timelocks and the time-value of money. When writing wallet and Lightning Network application software, it is necessary to ensure that sufficient time is given and users are able to have their transactions enter into the blockchain when interacting with non-cooperative or malicious channel counterparties.

9.2 Forced Expiration Spam

Forced expiration of many transactions may be the greatest systemic risk when using the Lightning Network. If a malicious participant creates many channels and forces them all to expire at once, these may overwhelm block data capacity, forcing expiration and broadcast to the blockchain. The result would be mass spam on the bitcoin network. The spam may delay transactions to the point where other locktimed transactions become valid.

This may be mitigated by permitting one transaction replacement on all pending transactions. Anti-spam can be used by permitting only one transaction replacement of a higher sequence number by the inverse of an even or odd number. For example, if an odd sequence number was broadcast, permit a replacement to a higher even number only once. Transactions would use the sequence number in an orderly way to replace other transactions. This mitigates the risk assuming honest miners. This attack is extremely high risk, as incorrect broadcast of Commitment Transactions entail a full penalty of all funds in the channel.

Additionally, one may attempt to steal HTLC transactions by forcing a timeout transaction to go through when it should not. This can be easily mitigated by having each transfer inside the channel be lower than the total transaction fees used. Since transactions are extremely cheap and do not hit the blockchain with cooperative channel counterparties, large transfers of value can be split into many small transfers. This attempt can only work if the blocks are completely full for a long time. While it is possible to mitigate it using a longer HTLC timeout duration, variable block sizes may become common, which may need mitigations.

If this type of transaction becomes the dominant form of transactions which are included on the blockchain, it may become necessary to increase the block size and run a variable blocksize structure and timestop flags as described in the section below. This can create sufficient penalties and disincentives to be highly unprofitable and unsuccessful for attackers, as attackers lose all their funds from broadcasting the wrong transaction, to the point where it will never occur.

9.3 Coin Theft via Cracking

As parties must be online and using private keys to sign, there is a possibility that, if the computer where the private keys are stored is compromised, coins will be stolen by the attacker. While there may be methods to mitigate the threat for the sender and the receiver, the intermediary nodes must be online and will likely be processing the transaction automatically. For this reason, the intermediary nodes will be at risk and should not be holding a substantial amount of money in this "hot wallet." Intermediary nodes which have better security will likely be able to out-compete others in the long run and be able to conduct greater transaction volume due to lower fees. Historically, one of the largest component of fees and interest in the financial system are from various forms of counterparty risk – in Bitcoin it is possible that the largest component in fees will be derived from security risk premiums.

A Funding Transaction may have multiple outputs with multiple Commitment Transactions, with the Funding Transaction key and some Commitment Transactions keys stored offline. It is possible to create an equivalent of a "Checking Account" and "Savings Account" by moving funds between outputs from a Funding Transaction, with the "Savings Account" stored offline and requiring additional signatures from security services.

9.4 Data Loss

When one party loses data, it is possible for the counterparty to steal funds. This can be mitigated by having a third party data storage service where encrypted data gets sent to this third party service which the party cannot decrypt. Additionally, one should choose channel counterparties who are responsible and willing to provide the current state, with some periodic tests of honesty.

9.5 Forgetting to Broadcast the Transaction in Time

If one does not broadcast a transaction at the correct time, the counterparty may steal funds. This can be mitigated by having a designated third party to send funds. An output fee can be added to create an incentive for this third party to watch the network. Further, this can also be mitigated by implementing OP_CHECKSEQUENCEVERIFY.

9.6 Inability to Make Necessary Soft-Forks

Changes are necessary to bitcoin, such as the malleability soft-fork. Additionally, if this system becomes popular, it will be necessary for the system to securely transact with many users and some kind of structure like a blockheight timestop will be desirable. This system assumes such changes to enable Lightning Network to exist entirely, as well as soft-forks ensuring the security is robust against attackers will occur. While the system may continue to operate with only some time lock and malleability soft-forks, there will be necessary soft-forks regarding systemic risks. Without proper community foresight, an inability to establish a timestop or similar function will allow systemic attacks to take place and may not be recognized as imperative until an attack actually occurs.

9.7 Colluding Miner Attacks

Miners may elect to refuse to enter in particular transactions (e.g. Breach Remedy transactions) in order to assist in timeout coin theft. An attacker can pay off all miners to refuse to include certain transactions in their mempool and blocks. The miners can identify their own blocks in an attempt to prove their behavior to the paying attacker.

This can be mitigated by encouraging miners to avoid identifying their own blocks. Further, it should be expected that this kind of payment to miners is malicious activity and the contract is unenforcible. Miners may then take payment and surreptitiously mine a block without identifying the block to the attacker. Since the attacker is paying for this, they will quickly run out of money by losing the fee to the miner, as well as losing all their money in the channel. This attack is unlikely and fairly unattractive as it is far too difficult and requires a high degree of collusion with extreme risk.

The risk model of this attack occurirng is similar to that of miners colluding to do reorg attacks: Extremely unlikely with many uncoordinated miners.

10 Block Size Increases and Consensus

If we presume that a decentralized payment network exists and one user will make 3 blockchain transactions per year on average, Bitcoin will be able to support over 35 million users with 1MB blocks in ideal circumstances (assuming 2000 transactions/MB, or 500 bytes/Tx). This is quite limited, and an increase of the block size may be necessary to support everyone in the world using Bitcoin. A simple increase of the block size would be a hard fork, meaning all nodes will need to update their wallets if they wish to participate in the network with the larger blocks.

While it may appear as though this system will mitigate the block size increases in the short term, if it achieves global scale, it will necessitate a block size increase in the long term. Creating a credible tool to help prevent blockchain spam designed to encourage transactions to timeout becomes imperative.

To mitigate timelock spam vulnerabilities, non-miner and miners' consensus rules may also differ if the miners' consensus rules are more restrictive. Non-miners may accept blocks over 1MB, while miners may have different soft-caps on block sizes. If a block size is above that cap, then that is viewed as an invalid block by other miners, but not by non-miners. The miners will only build the chain on blocks which are valid according to the agreed-upon soft-cap. This permits miners to agree on raising the block size limit without requiring frequent hard-forks from clients, so long as the amount raised by miners does not go over the clients' hard limit. This mitigates the risk of mass expiry of transactions at once. All transactions which are not redeemed via Exercise Settlement (ES) may have a very high fee attached, and miners may use a consensus rule whereby those transactions are exempted from the soft-cap, making it very likely the correct transactions will enter the blockchain.

When transactions are viewed as circuits and contracts instead of transaction packets, the consensus risks can be measured by the amount of time available to cover the UTXO set controlled by hostile parties. In effect, the upper bound of the UTXO size is determined by transaction fees and the standard minimum transaction output value. If the bitcoin miners have a deterministic mempool which prioritizes transactions respecting a "weak" local time order of transactions, it could become extremely unprofitable and unlikely for an attack to succeed. Any transaction spam time attack by broadcasting the incorrect Commitment Transaction is extremely high risk for the attacker, as it requires an immense amount of bitcoin and all funds committed in those transactions will be lost if the attacker fails.

11 Use Cases

In addition to helping bitcoin scale, there are many uses for transactions on the Lightning Network:

- Instant Transactions. Using Lightning, Bitcoin transactions are now nearly instant with any party. It is possible to pay for a cup of coffee with direct non-revocable payment in milliseconds to seconds.

- Exchange Arbitrage. There is presently incentive to hold funds on exchanges to be ready for large market moves due to 3-6 block confirmation times. It is possible for the exchange to participate in this network and for clients to move their funds on and off the exchange for orders nearly instantly. If the exchange does not have deep market depth and commits to only permitting limit orders close to the top of the order book, then the risk of coin theft becomes much lower. The exchange, in effect, would no longer have any need for a cold storage wallet. This may substantially reduce thefts and the need for trusted third party custodians.

- Micropayments. Bitcoin blockchain fees are far too high to accept micropayments, especially with the smallest of values. With this system, near-instant micropayments using Bitcoin without a 3rd party custodian would be possible. It would enable, for example, paying per-megabyte for internet service or per-article to read a newspaper.

- Financial Smart Contracts and Escrow. Financial contracts are especially time-sensitive and have higher demands on blockchain computation. By moving the overwhelming majority of trustless transactions off-chain, it is possible to have highly complex transaction contract terms without ever hitting the blockchain.

- Cross-Chain Payments. So long as there are similar hash-functions across chains, it's possible for transactions to be routed over multiple chains with different consensus rules. The sender does not have to trust or even know about the other chains – even the destination chain. Simiarly, the receiver does not have to know anything about the sender's chain or any other chain. All the receiver cares about is a conditional payment upon knowledge of a secret on their chain.

Payment can be routed by participants in both chains in the hop. E.g. Alice is on Bitcoin, Bob is on both Bitcoin and X-Coin and Carol is on a hypothetical X-Coin, Alice can pay Carol without understanding the X-Coin consensus rules.

12 Conclusion

Creating a network of micropayment channels enables bitcoin scalability, micropayments down to the satoshi, and near-instant transactions. These channels represent real Bitcoin transactions, using the Bitcoin scripting opcodes to enable the transfer of funds without risk of counterparty theft, especially with long-term miner risk mitigations.

If all transactions using Bitcoin were on the blockchain, to enable 7 billion people to make two transactions per day, it would require 24GB blocks every ten minutes at best (presuming 250 bytes per transaction and 144 blocks per day). Conducting all global payment transactions on the blockchain today implies miners will need to do an incredible amount of computation, severely limiting bitcoin scalability and full nodes to a few centralized processors.

If all transactions using Bitcoin were conducted inside a network of micropayment channels, to enable 7 billion people to make two channels per year with unlimited transactions inside the channel, it would require 133 MB blocks (presuming 500 bytes per transaction and 52560 blocks per year). Current generation desktop computers will be able to run a full node with old blocks pruned out on 2TB of storage.

With a network of instantly confirmed micropayment channels whose payments are encumbered by timelocks and hashlock outputs, Bitcoin can scale to billions of users without custodial risk or blockchain centralization when transactions are conducted securely off-chain using bitcoin scripting, with enforcement of non-cooperation by broadcasting signed multisignature transactions on the blockchain.

13 Acknowledgements

Micropayment channels have been developed by many parties, and has been discussed on bitcointalk, the bitcoin mailing list, and IRC. The amount of

contributors to this idea are immense and much thought have been put into this ability. Effort has been placed into citing and finding similar ideas, however it is absolutely not near complete. In particular, there are many similarities to a proposal by Alex Akselrod by using hashlocking as a method of encumbering a hub-and-spoke payment channel.

Thanks to Peter Todd for correcting a significant error in the HTLC script, as well as optimizing the opcode size.

Thanks to Elizabeth Stark for reviewing and corrections.

Thanks to Rusty Russell for reviewing this document and suggestions for making the concept more digestible, as well as working on a construction which may provide a stop-gap solution before a long-term malleability fix (to be described in a future version).

Appendix A Resolving Malleability

In order to create these contracts in Bitcoin without a third party trusted service, Bitcoin must fix the transaction malleability problem. If transactions can be mutated, then signatures can be invalidated, thereby making refund transactions and commitment bonds invalidated. This creates an opportunity for hostile actors to use it as an opportunity for a negotiating tactic to steal coins, in effect, a hostage scenario.

To mitigate malleability, it is necessary to make a soft-fork change to bitcoin. Older clients would still work, but miners would need to update. Bitcoin has had several soft forks in the past, including pay-to-script-hash (P2SH).

To mitigate malleability, it requires changing which contents are signed by the participants. This is achieved by creating new sighash types. In order to accommodate this new behavior, a new P2SH type or new OP_CHECKSIG is necessary to make it a soft-fork rather than a hard-fork.

If a new P2SH was defined, it would use a different output script such as:

```
OP_DUP OP_HASH160 <20-byte hash> OP_EQUALVERIFY
```

Since this will always resolve to true provided a valid redeemScript, all existing clients will return true. This allows the scripting system to construct new rules, including new signature validation rules. At least one new sighash would need to exist.

SIGHASH_NOINPUT would neither sign any input transactions IDs nor sign the index. By using SIGHASH_NOINPUT, one can be assured that one's counterparty cannot invalidate entire trees of chained transactions of potential contract states which were previously agreed upon, using transaction ID mutation. With the new sighash flags, it is possible to spend from a parent transaction even though the transaction ID has changed, so long as the script evaluates as true (i.e. a valid signature).

SIGHASH_NOINPUT implies significant risk with address reuse, as it can work with any transaction in which the sigScript returns as valid, so multiple transactions with the same outputs are redeemable (provided the output values are less).

Further, and just as importantly, SIGHASH_NOINPUT permits participants to sign spends of transactions without knowing the signatures of the transaction being spent. By solving malleability in the above manner, two parties may build contracts and spend transactions without either party having the ability to broadcast that original transaction on the blockchain until both parties agree. With the new sighash type, participants may build potential contract states and potential payout conditions and agree upon all terms, before the contract may be paid, broadcast, and executed upon without the need for a trusted third party.

Without SIGHASH_NOINPUT, one cannot build outputs before the transaction can be funded. It is as if one cannot make any agreements without committing funds without knowing what one is committing to. SIGHASH_NOINPUT allows one to build redemption for transactions which do not yet exist. In other words, one can form agreements before funding the transaction if the output is a 2-of-2 multisignature transaction.

To use SIGHASH_NOINPUT, one builds a Funding Transaction, and does not yet sign it. This Funding Transaction does not need to use SIGHASH_NOINPUT if it is spending from a transaction which has already been entered into the blockchain. To spend from a Funding Transaction with a 2-of-2 multisignature output which has not yet been signed and broadcast, however, requires using SIGHASH_NOINPUT.

A further stop-gap solution using OP_CHECKSEQUENCEVERIFY or a less-optimal use of OP_CHECKLOCKTIMEVERIFY will be described in a future paper by Rusty Russell. An updated version of this paper will also include these constructions.

References

[1] Satoshi Nakamoto. Bitcoin: A Peer-to-peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, Oct 2008.

[2] Manny Trillo. Stress Test Prepares VisaNet for the Most Wonderful Time of the Year. http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html, Oct 2013.

[3] Bitcoin Wiki. Contracts: Example 7: Rapidly-adjusted (micro)payments to a pre-determined party. https://en.bitcoin.it/wiki/Contracts#Example_7:_Rapidly-adjusted_.28micro.29payments_to_a_pre-determined_party.

[4] bitcoinj. Working with micropayment channels. https://bitcoinj.github.io/working-with-micropayments.

[5] Leslie Lamport. The Part-Time Parliament. *ACM Transactions on Computer Systems*, 21(2):133–169, May 1998.

[6] Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, Jul 1978.

[7] Alex Akselrod. Draft. https://en.bitcoin.it/wiki/User:Aakselrod/Draft, Mar 2013.

[8] Alex Akselrod. ESCHATON. https://gist.github.com/aakselrod/9964667, Apr 2014.

[9] Peter Todd. Near-zero fee transactions with hub-and-spoke micropayments. http://sourceforge.net/p/bitcoin/mailman/message/33144746/, Dec 2014.

[10] C.J. Plooy. Combining Bitcoin and the Ripple to create a fast, scalable, decentralized, anonymous, low-trust payment network. http://www.ultimatestunts.nl/bitcoin/ripple_bitcoin_draft_2.pdf, Jan 2013.

[11] BitPay. Impulse. http://impulse.is/impulse.pdf, Jan 2015.

[12] Mark Friedenbach. BIP 0068: Consensus-enforced transaction replacement signaled via sequence numbers (relative locktime). https://github.com/bitcoin/bips/blob/master/bip-0068.mediawiki, May 2015.

[13] Mark Friedenbach BtcDrak and Eric Lombrozo. BIP 0112: CHECKSEQUENCEVERIFY. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki, Aug 2015.

[14] Jonas Schnelli. What does OP_CHECKSEQUENCEVERIFY do? http://bitcoin.stackexchange.com/a/38846, Jul 2015.

[15] Greg Maxwell (nullc). reddit. https://www.reddit.com/r/Bitcoin/comments/37fxqd/it_looks_like_blockstream_is_working_on_the/crmr5p2, May 2015.

[16] Gavin Andresen. BIP 0016: Pay to Script Hash. https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki, Jan 2012.

[17] Pieter Wuille. BIP 0032: Hierarchical Deterministic Wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki, Feb 2012.

[18] Ilja Gerhardt and Timo Hanke. Homomorphic Payment Addresses and the Pay-to-Contract Protocol. http://arxiv.org/abs/1212.3257, Dec 2012.

[19] Nick Szabo. Formalizing and Securing Relationships on Public Networks. http://szabo.best.vwh.net/formalize.html, Sep 1997.