Discreet Log Contracts Channels and Integration in the Lightning Network

Ichiro Kuwahara
Crypto Garage
Tokyo, Japan
kuwahara@cryptogarage.co.jp

Thibaut Le Guilly
Crypto Garage
Tokyo, Japan
thibaut@cryptogarage.co.jp

Takatoshi Nakagawa
DG Lab
Tokyo, Japan
nakagat@gmail.com

***Abstract*—Contracts established on a blockchain remove the need for intermediary third parties, but usually require external data to decide on an outcome, provided by a so-called oracle. Discreet Log Contracts (DLC) enable establishing such contracts directly on the Bitcoin blockchain without direct interaction with an oracle, increasing the privacy of the contracting parties. Using the Bitcoin blockchain however also means hitting the scalability issues that are inherent to it. To solve this issue, we propose a construction for establishing DLC off-chain through channels enabling the execution of consecutive contracts requiring only interaction with the blockchain during the setup and closing phases, similar to the functioning of the Lightning Network. We provide an analysis of the construction to ensure its security, and then describe how DLC channels could be integrated in parallel to Lightning Network channels making them easier to use and enabling reuse of the Lightning Network's infrastructure.**

***Index Terms*—Bitcoin, smart contract, payment channels**

1. Introduction

More than ten years after the writing of the Bitcoin white paper [12], the Bitcoin blockchain is not only used as a way to transfer value, but also to establish financial contracts in a decentralized way. These contracts can be specified using a simple scripting language (called Script in Bitcoin) that describes the spending conditions in a transaction. They however usually rely on external data, such as a currency exchange rate, that is not directly available within the execution environment of the scripts. To work around this limitation, a solution is to use oracles [9], which provide such information in a format that can be used within a script's execution environment (usually in the form of a digital signature). While putting a minimum amount of trust into oracles is unavoidable (assuming that they will not collude with one of the party in the contract), having the parties requesting data to them leaks the existence of the contract.

Discreet Log Contracts (DLC) [8] aim to solve this issue. They enable two parties who do not trust each others to establish financial contracts directly on the Bitcoin blockchain without requiring direct interaction with an oracle. This indirect interaction increases the privacy of the contracting parties as the oracle need not be aware of the contract's existence. DLC are also peer to peer contracts, in that they don't require any intermediate party to be established. As an example, with DLC, futures and forward contracts can be established between two parties without having to put any of the funds in the custody of a cryptocurrency exchange. However, as DLC are settled on the Bitcoin blockchain, the settlement time as well as the fees that need to be paid to the miners can limit its scalability [5]. This puts DLC at a disadvantage compared to centralized solutions.

A common solution to these limitations is to make use of second layer protocols [10], that enable the transfer of assets between parties outside of the blockchain (thus often called off-chain protocols), while still taking advantage of its security properties. Payment channels, and in particular the Lightning Network [15], is currently one of the most widely used layer two solution. We recall the building blocks of payment channels, as well as the functioning of DLC in Section 2. Inspired by payment channels, we present in Section 3 a construction for establishing DLC channels, enabling the execution of consecutive contracts requiring interaction with the blockchain only during the setup and closing phases.

In order to implement these channels, a lot of work has to be done at multiple layers. At the network level, the creation of a peer to peer network is a tedious task. At the application level, monitoring of the blockchain for the broadcast of transactions from revoked states, as well as ensuring that liquidity is available in the channels are non trivial issues to be solved. Fortunately, solutions have already been proposed in the context of the Lightning Network. Having the ability for DLC channels to be embedded into the Lightning Network infrastructure would thus be highly beneficial. Section 4 describes an approach to achieve this.

While the constructions presented in this paper have to the best of our knowledge not been proposed before, they draw inspiration from past and related work that we discuss in Section 5. We finally conclude and provide directions for further work in Section 6.

2. Preliminaries

This section introduces the basic building blocks of payment channels as well as the original concept of DLC.

2.1. Transactions

In Bitcoin a transaction consists of a number of inputs and outputs. Each output includes a script which specifies

under which conditions the ownership of the value contained inside it can be transferred. Each input refers to a previous output, and needs to provide arguments to satisfy its spending conditions. Common examples are outputs checking for a digital signature from a specific private key, or requiring signatures from *m* out of *n* possible keys. Outputs can also contain multiple spending conditions, or spending paths (for example one requiring a single signature, and another one requiring two) that can be created using `IF/ELSE` statements.

2.2. Payment channels

Payment channels are currently among the most popular solutions to increasing the scalability of blockchain systems. The concept of payment channels usually revolves around broadcasting a limited number of transactions on the blockchain (called *on-chain* transactions), while enabling a much larger number of transactions to occur. This can be securely achieved by having two parties exchange transactions and digital signatures in such a way that it is always possible for both of them to enforce the latest state of the channel (the balance between both parties) by broadcasting a transaction *on-chain*, as well as to recover their funds in case their counter party broadcast an outdated state. A basic assumption for payment channel is that both parties must be watching the blockchain for inclusion of stale transactions, or at least delegate this task to watchtowers [11].

2.3. Transaction revocation

In order to protect the parties from the broadcast of an outdated state of the channel, the Lightning Network uses a transaction revocation mechanism. It works by having both parties hold a different transaction with the same output values, but where the output paying them contains two spending paths: * The first one requires a signature from a *revocation key*, * The second one requires a signature from the party that holds it, but can only be spent after a certain amount of time has elapsed (using the `CHECKSEQUENCEVERIFY` opcode [3]).

A party can revoke a transaction by revealing the private key for the first spending path. If this transaction was to be broadcast *on-chain* in the future, the counter party has a certain amount of time to use the revocation key to claim the funds from the output. Of course the parties should never reveal their signatures for *their* revokable transactions, as it could allow their counter party to steal their fund if they also know the revocation key.

Note that the protocols and mechanisms for transactions and key exchanges are kept simple in this paper for the purpose of explanation, but the interested reader can refer to the Lightning Network specifications ([13], [14]) for examples of how it can be done in practice.

2.4. Discreet Log Contracts

DLC were proposed to enable two parties to enter into a financial contract using Bitcoin transactions and an oracle, but without requiring direct interaction with the oracle, thus leaving a “discreet log”. In other words, the parties do not have to inform the oracle of the contract being established. This is achieved through a peculiar use of Schnorr signatures [16] that is described in detail in [8].

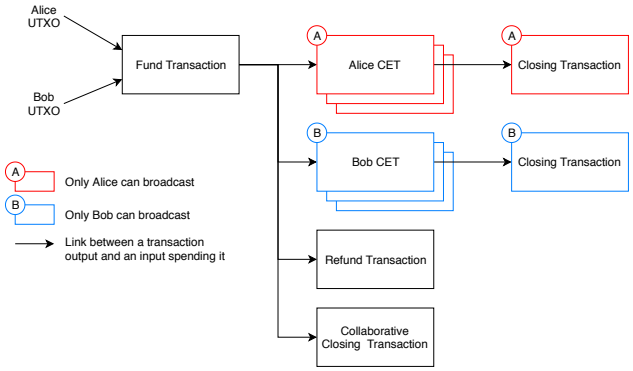

Figure 1. Illustration of the on-chain DLC transaction flow. Both parties lock their collaterals in the fund transaction, and can exercise the outcome of the contract through their CETs. Alternatively, the parties can agree to close the contract collaboratively, and if the oracle fails to produce a signature, they can recover their collaterals with the refund transaction.

The second important property of DLC is that they don’t require trust between the parties, and only minimal trust to the oracle (essentially that it does not collude with one of the party). This is achieved through the use of specially crafted Bitcoin transactions that we recall in this section. The transaction flow is illustrated in Figure 1.

2.5. Fund transaction

The first transaction that makes up a DLC is the fund transaction. This transaction takes a number of inputs coming from each of the parties UTXOs and locks the collateral of each party into a multi-signature script, requiring a signature from both parties to be unlocked. It can also include change outputs when the sum of one of the party’s input UTXOs is greater than the desired contract collateral. The output script for the multi-signature output of the fund transaction is detailed in Script 1.

2.6. Refund Transaction

The refund transaction is one of the transactions that can be used to spend from the fund transaction, and simply returns the collateral posted by each party. It is intended to be used only in the case where the oracle does not publish a signature for the event on which the DLC is based on. For that reason, it is time locked, meaning that it cannot be included in the blockchain before a certain timestamp, that should be set to after the maturity date of the contract.

2.7. Contract Execution Transactions

A contract execution transaction (CET) is the second type of transaction that can spend from the fund transaction. It encodes a possible outcome of the contract, meaning that there must be as many CETs as possible outcomes of the contract. Both parties hold a different set of CETs, each including two outputs.

The first output of a CET consists of a script that enables either the party broadcasting it to spend it by using a combination of its private key and the oracle signature, or their counter party after a certain time has elapsed after the transaction was included in the blockchain (using the Check Sequence Verify op-code [3], the script is detailed in Script 5). This ensures that if one of the parties was to broadcast a CET that did not correspond to the outcome revealed by the oracle, they would lose their fund (this is somewhat similar to the transaction revocation concept discussed in Section 2.3).

The second output can be spent directly by the counter party of the broadcaster.

2.8. Closing Transaction

The closing transaction is used by the broadcaster of a CET to retrieve the funds locked in the first output. It satisfies the script of the first CET output by providing a signature that is created using a combination of the spender's private key and the oracle signature.

2.9. Signing Order

In order to guarantee that both parties can always either recover their funds or execute the contract, the transactions need to be signed in a specific order.

First, the signatures for the CETs and the refund transaction are exchanged. As the fund transaction they spend from has not been signed yet, they cannot be used.

The signatures for the fund transaction are then exchanged. Note that as this exchange happens non atomically, one of the parties will obtain the other party's signature first. This gives them a "free option", in the sense that they can choose to execute the contract only in the event that the outcome is favorable to them. However, if the counter party does not receive the expected signature, they can choose to spend their UTXOs included in the fund transaction, rendering it invalid, and thus cancelling the contract.

2.10. Collaborative Closing

Given the above described transactions, executing a DLC requires broadcasting three transactions: the fund transaction, a CET and a closing transaction. However, once the Oracle has published the outcome, the parties can decide to create a collaborative closing transaction with the same output amounts as the CET for the announced outcome. This reduces the number of required transactions to two. If the parties would like to re-establish a contract after maturity, another two transactions will have to be broadcast *on-chain*.

3. DLC Channels

In this section, we first describe a naive approach to establishing and updating a DLC contract off-chain, and show why it fails. We then present our solution to securely do it.

3.1. A straw-man proposal

A simple approach to establish off-chain DLC is to make CETs and the refund transaction revocable. We assume two parties, Alice and Bob, want to create a DLC channel, to be able to establish multiple consecutive contracts. The protocol would be as follow:

1) Alice and Bob create all transactions and exchange signatures according to the *regular* DLC protocol,
2) Bob broadcasts the fund transaction,
3) After contract maturity, they create a new set of CETs and a new refund transaction representing the new contract to establish,
4) Alice sends to Bob her signatures for the new CETs and refund transaction,
5) Bob sends to Alice his signatures for the new transactions,
6) Alice sends to Bob her revocation key, *revoking* her previous CETs and refund transaction,
7) Bob sends to Alice his revocation key for the previous DLC transactions.

At the end of this protocol, both Alice and Bob have the set of signed transactions for the second DLC, and the transactions for the previous one are revoked. However, there is an issue at step 6. After sending her revocation secret to Bob, Alice cannot anymore enforce the result from the first contract. However, as Bob has not yet revealed his secret, he still has the ability to do so, in addition to having the ability to enforce the second one. If Bob is dishonest, he could thus choose not to reveal his revocation key, and wait until just before the maturity of the second contract to execute the one that is most favorable to him (he could also potentially use the previous refund transaction if the time lock expired). Note that the Lightning Network does not suffer from this issue, as when updating a channel, the payer should reveal their secret first as the update will be unfavorable to them.

In order to solve this issue, we introduce two new types of transactions in the protocol.

3.2. Update transaction

The first new transaction is the update transaction. It is similar in structure to the *commitment* transaction in the Lightning Network, taking as input the multi-signature output of the fund transaction, and containing two outputs, representing the current balance of each party in the channel. Each party holds a different update transaction, in order to enable their revocation. The first output contains three different spending paths (the script for this output is detailed in Script 2). We describe here the required arguments for the three paths of the first output of the transaction held by Alice:

1) A signature from Alice's revocation key,
2) A signature from both Alice and Bob, plus a time delay (CSV) d_1,
3) A signature from Alice, plus a time delay (CSV) d_2.

The second output contains two different spending paths (the script is detailed in Script 3), requiring either:

1) A signature from both Alice and Bob,
2) A signature from Bob, plus a time delay (CSV) d_2.

Note that it is necessary that $d_1 < d_2$ as will become clear in the description of the protocol in Section 3.4. The version held by Bob is similar, with Bob and Alice's signatures reversed.

3.3. Buffer transaction

A buffer transaction consumes both outputs of an update transaction, using the multi-signature paths. It combines the value of these inputs into a single output, requiring both Alice and Bob signatures to be unlocked (the script is detailed in Script 4).

3.4. Offchain DLC protocol

Having these two new types of transactions, we can describe the protocol for establishing and updating a DLC channel. The transaction flow is illustrated in Figure 2.

3.4.1. Establishment. The establishment of a DLC channel is similar to that of an *on-chain* DLC. We make use of the two new types of transaction for consistency across the protocol steps, but note that at this stage they are not strictly required (one could instead make the CETs and the refund transaction revokable for this initial contract, as long as the update and buffer transactions are used for establishing the second one). In the following, we assume that the terms of the contract are already agreed upon between the participants:

1) Alice sends to Bob the set of UTXOs she wants to use to fund the channel, a public key for later revocation, as well as a set of public keys to be used in the transactions,
2) Bob selects a set of UTXOs, and using the information received from Alice constructs the transactions and generates signatures for them,
3) Bob sends back his set of UTXOs and public keys, as well as his signatures for all transactions except for the fund transaction and *his* update transaction,
4) Alice creates the transactions and signatures, and sends the signatures to Bob (except for *her* update transaction),
5) Bob adds the signatures to the fund transaction and broadcasts it.

As in regular DLC, parties are ensured that their funds can always be unlocked by signing the fund transaction last. Note that both parties should validate the signatures they receive.

3.4.2. Update. At contract maturity, once the oracle has released a signature, Alice and Bob can decide to create a mutual closing transaction to terminate the contract, or if one party is uncooperative, the other party can broadcast the update, buffer, appropriate CET and closing transactions. We assume here that instead they wish to enter in another contract, and have already agreed on the terms. They can then use the following protocol:

1) Alice and Bob generate the set of transactions (update, buffer, CETs and refund) for the new contract, with the update transactions having output values equal to the outcome of the previous contract,
2) Alice sends her signatures for Bob's update transaction, all CETs and the refund transactions,
3) Bob sends his signatures for Alice's update transaction, all CETs, the refund transactions, as well as the revocation key for his previous update transaction,
4) Alice sends the revocation key for her previous update transaction, as well as the signature for her buffer transaction,
5) Bob sends his signatures for both buffer transactions.
6) Alice sends her signature for Bob's buffer transaction.

We go through the different steps of this protocol to analyze the possible actions of both participants after each step.

Step 2. : Bob obtained Alice's signature for his update transaction, giving him the ability to broadcast it. However, as he does not have Alice's signature for the following buffer transaction, he would only be able to use the third script path returning him the same amount as decided by the outcome of the previous contract, after expiration of the time lock. Alice would also recover the same amount as she was awarded by the outcome of the previous contract. Given the information held by each party at this stage, Bob's update transaction resembles a mutual closing transaction, with a time delay. Alice on her side still has the possibility of unilaterally closing the previous contract would Bob stop cooperating.

Step 3. : Bob has revealed the revocation key for his previous update transaction, and can thus no longer broadcast it safely without risking to be penalized. But since he has the ability to recover his due funds with the last update transaction, he still can exit the channel would Alice stop cooperating. Alice also obtained the ability to broadcast her update transaction (but not the following buffer transaction).

Step 4. : Alice revoked her previous update transaction, and Bob has the ability to broadcast Alice's buffer transactions. That means that if Alice were to broadcast her fund transaction, Bob can force Alice to honor the contract. While Alice doesn't yet have the same ability, broadcast of Bob's update transaction would only lead to settling the channel on the previous contract's outcome. If Bob were to stop collaborating at this point, Alice can broadcast her update transaction to either force Bob to broadcast her buffer transaction and enter in the contract, or exit the contract at the latest state.

Step 5. : Alice obtained the ability to broadcast both buffer transactions, either for unilaterally closing the contract after maturity using the path starting with her update transaction, or for forcing Bob to honor the contract were he to broadcast his update transaction before the term.

Step 6. : Bob now also has the ability to broadcast all transactions, except for Alice's update transaction.

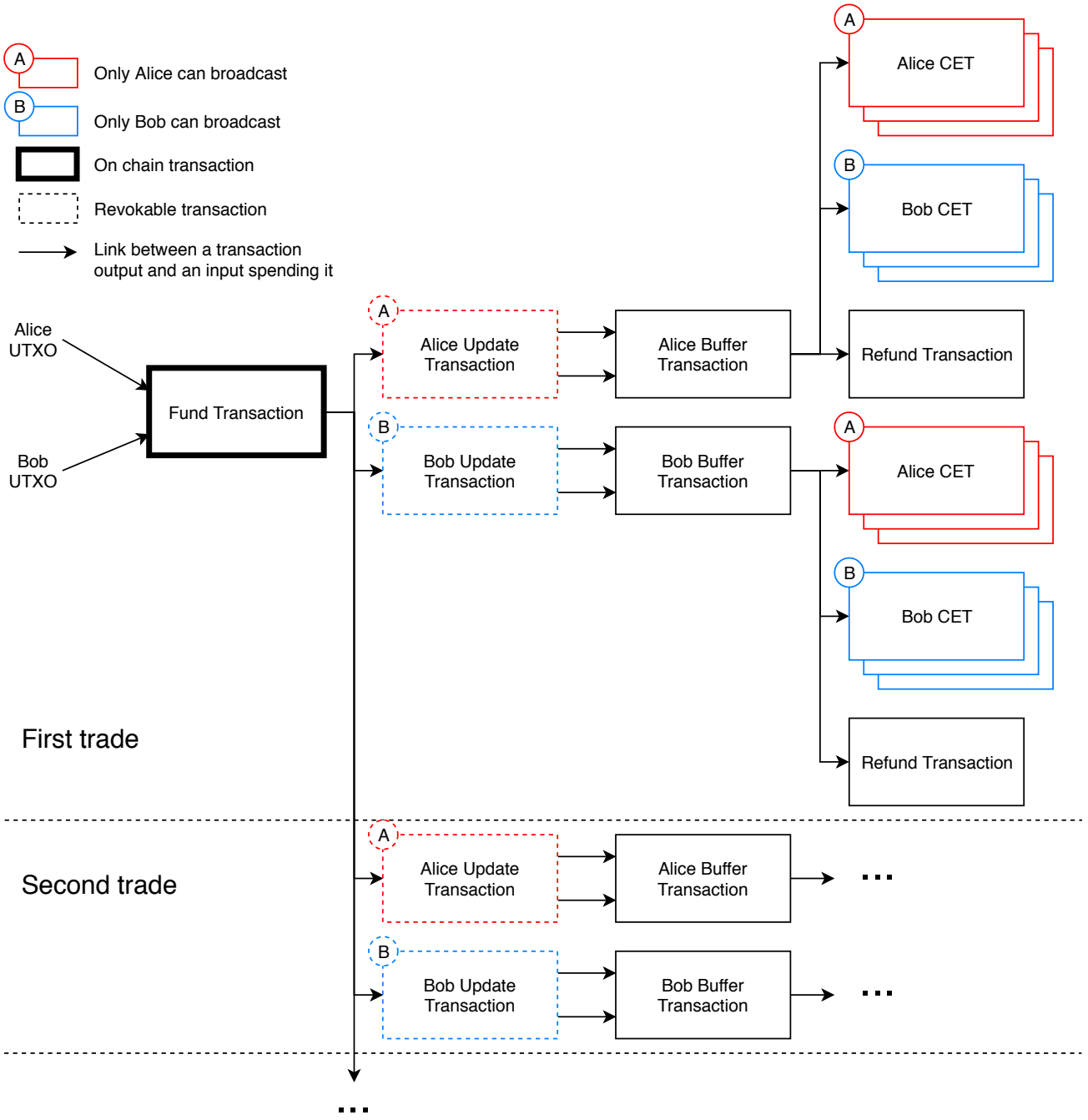

Figure 2. Illustration of the transaction flow for the establishment and update of a DLC channel. The addition of the update and buffer transactions enables both parties to safely re-enter in a contract after a previous one has expired.

3.5. Comparison with on-chain protocol

Compared with the on-chain version, we note that the off-chain protocol leads to a larger number of transactions to be broadcast in the case of a unilateral close (5 compared to 3). However, unilateral closes are expected to be an exception, and cooperative closes still only require two transactions to be broadcast. In addition, the off-chain protocol enables the execution of an unlimited number of contracts, and apart from the update and buffer transactions, the other transactions are the same as the on-chain version, meaning that implementations can reuse the code from the off-chain version.

4. Integration with the Lightning Network

While having the protocol presented in Section 3.4 theoretically enables the creation of DLC channels, implementing all the necessary infrastructure would require a lot of additional work. A lot of this work has already been carried on by the many researchers and developers who have contributed to the development of the Lightning Network. Leveraging this work, and new or on-going research such as channel splicing, sub-marine swaps or channel factories [4] for the creation of DLC channels would thus reduce the required effort.

In addition, if two parties who already have an estab-

lished Lightning Network channel open with each others wished to create a DLC channel, it would be impractical and wasteful for them to have to broadcast another fund transaction. Another issue arises if the parties wish to establish several contracts concurrently, as this would require opening several channels between the two parties. Finally, as two parties participate in consecutive contracts within a DLC channel, it might become imbalanced, preventing further contracts to take place.

In this section, we propose a way to integrate DLC channels within the Lightning Network to enable: * Reusing the Lightning Network infrastructure for peer discovery, channel funding, as well as recent innovations such as channel factories or splicing, * the two parties of a lightning channel to establish multiple DLC channels in parallel to the lightning one, reusing the same fund transaction, * the reallocation of funds between a lightning channel and DLC channels.

An illustration of the transactions for the integration of a single DLC channel with a lightning one is given in Figure 3.

4.1. Split transaction

To divide the allocation of funds from the fund transaction to both the lightning and DLC channels, we introduce a *split transaction*. This transaction contains multiple outputs, one for the funding of the lightning channel, and one for the funding of each of the DLC channels, each with a revocation path and a multi signature path (detailed in Script 6). Due to the revocation paths, each party needs to hold a different version of a split transaction.

4.2. Channel splitting

Here we assume that a dual funded lightning channel already exists between Alice and Bob, that they wish to establish a single DLC channel in parallel (the protocol generalizes to multiple ones), and that they have agreed on how much funds to allocate to each channel. The protocol for a channel split is as follow:

1) Alice sends signatures for Bob's update (for the DLC channel), commitment (for the lightning channel) and split transactions,
2) Bob sends signatures for Alice's update, commitment and split transactions, as well as the secret for his previous commitment transaction,
3) Alice sends the secret to her previous commitment transaction,
4) Alice and Bob follow a protocol similar to presented in Section 3.4 to finish the establishment of the DLC channel, with the difference of having to handle four sets of transactions instead of two (due to the asymmetry of the *split transaction*).

Subsequently, both channels can be updated independently.

If Alice and Bob wish to re-balance the funds between the lightning and DLC channels, they can follow a similar protocol, but revoke the split transactions instead of the initial commitment transactions of the lightning channel.

Finally, they can also close the DLC channel and return to a simple lightning channel by creating new commitment transactions allocating the sums of the funds of each party in each channel before revoking the *split transaction*.

4.3. Discussion

One of the main drawback of our proposed split channel design, is the larger amount of transactions to keep track of. This increases the complexity of the system as well as the burden for watchtowers [11].

Another approach to integrate DLC channels together with lightning channel would be to use an extra output in the lightning channel's commitment transaction, that can be used as a funding input to a DLC. This was proposed by Bednár and Pickhardt [2]. While it does effectively reduce the number of transactions, it also means that for each update of the lightning channel, all the transactions for the DLC need to be reconstructed, signed and exchanged. The update of each channel also becomes dependent on the state of the other channel, potentially leading to concurrency issues. We thus believe that the extra cost induced by the split of the channel is compensated by a better separation between the two channels and the ability to update them independently.

We also note that this splitting approach, while designed in the context of allowing DLC channels within the Lightning network, could also be used to integrate other types of channel relying on a fund transaction.

5. Related Work

As already discussed in Section 4.3, Bednár and Pickhardt [2] previously proposed DLC channels embedded in the Lightning Network. Their construction works by adding an extra output to the commitment transaction of a lightning channel, which is used as the funding for a DLC. One of the disadvantage of this approach is the complex interaction between the DLC and lightning channels. They also mention splitting a channel into a DLC and lightning one by extending the peer protocol of the Lightning Network, but details on how this would be achieved are not given making it difficult to compare with our proposed approach. Finally, the authors also mention the possibility to improve the construction if Bitcoin introduces the `SIGHASH_NOINPUT` signature hash, as well as using channel factories. `SIGHASH_NOINPUT` would indeed allow for much simpler and more general constructions similar to the eltoo protocol [6].

Channel factories [4] could indeed prove useful to make it easier to create DLC channels as well as increase the scalability for channel setup.

While we chose the penalty model of the Lightning Network as a basis for the DLC channel construction, in part due to the desire of enabling integration within it, the model of micro-payment channels [7] could also be a viable approach, potentially reducing the asymmetry of the transactions.

6. Conclusion and Future Work

In this paper we presented a construction for the establishment of DLC channels. These channels enable

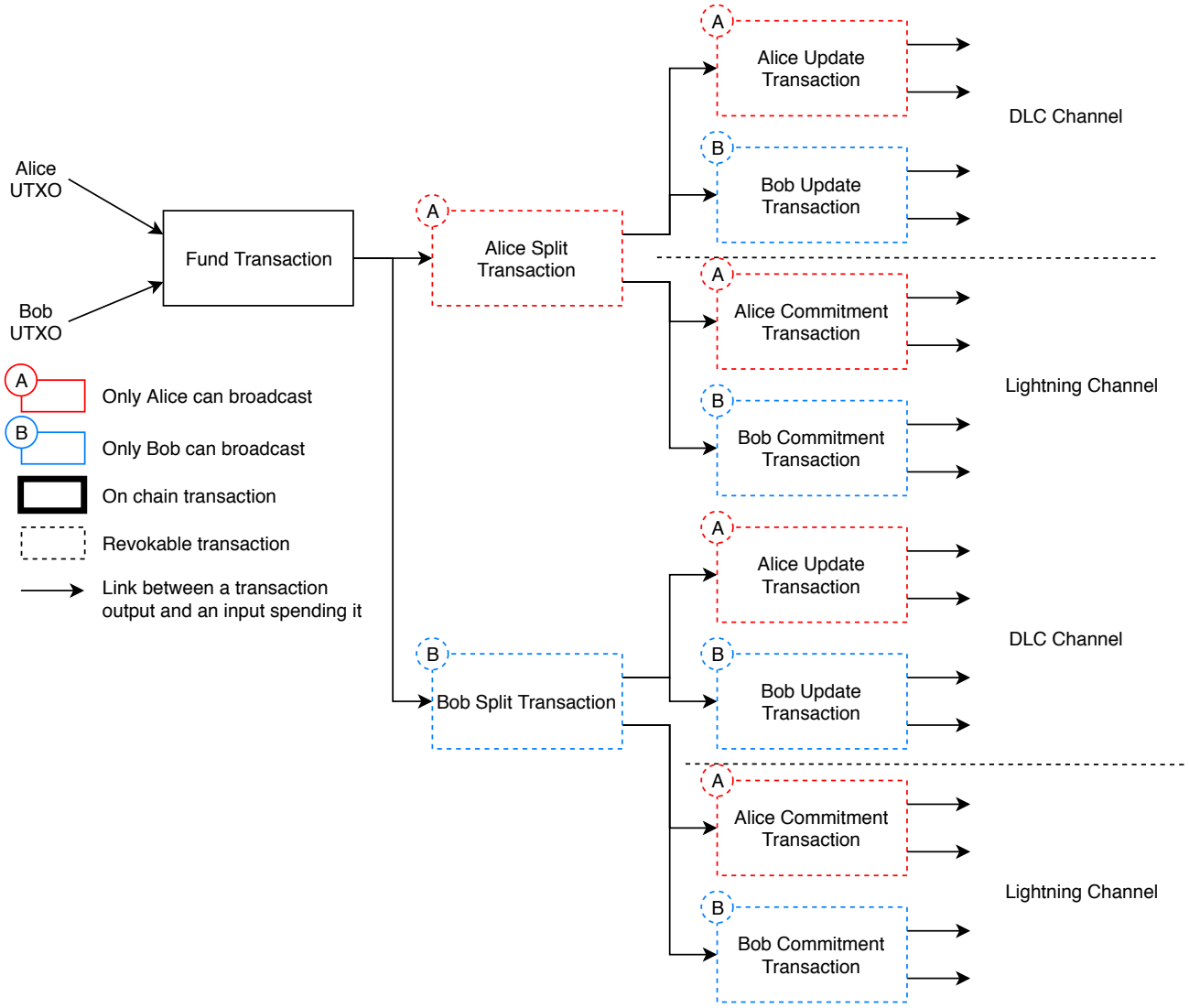

Figure 3. Illustration of the transaction construction for splitting a channel into a DLC and lightning one.

the execution of multiple consecutive contracts between two parties, while requiring the same amount of on-chain transactions as regular DLC in the best case, and two more transactions in the non-cooperative case. We then proposed an approach to integrate DLC channels within the Lightning Network, to reuse infrastructure and technological advances. This approach could also be used to integrate other types of protocols within the Lightning Network, as long as they use a fund transaction. We plan on implementing this integration in the near future, with a goal of enabling Lightning Network users to enter in various contracts with the parties with which they have a channel open.

Another important part of DLC not touched upon in this paper is the communication between the oracle and the contracting parties. Here again, leveraging the existing Lightning Network peer to peer network layer could reduce the amount of required work. Communication through the Lightning Network channels could also be considered, enabling payments to oracles.

Potential upgrades to the Bitcoin protocol will also have an impact on how DLC channels can be implemented. Firstly, the native support of Schnorr signatures will enable scriptless script versions of DLC. This would remove the necessity of a penalty mechanism, as well as enabling contracts between three or more parties to be established. Secondly, the addition of a signature hash type such as `SIGHASH_NOINPUT` would greatly reduce the complexity of the channel construction. We will thus be monitoring these advances closely to integrate them in the planned implementation of DLC channels when they become available.

Finally, while the analysis of the protocols presented in this paper gives us confidence in their correctness, we would like in future work to leverage the advances in formal analysis applied to Bitcoin [1] to formally verify them.

References

[1] Massimo Bartoletti and Roberto Zunino. Bitml: a calculus for bitcoin smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 83–100, 2018.

[2] Juraj Bednár and Pickhardt René. Lightning discreet log contract channels. https://hackmd.io/@lpQxZaCeTG6OJZI3awxQPQ/LN-DLC. Accessed: 2020-02-26.

[3] BtcDrak, Mark Friedenback, and Eric Lombrozo. Checksequenceverify. https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki, 2015. Accessed: 2020-02-26.

[4] Conrad Burchert, Christian Decker, and Roger Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science*, 5(8), 2018.

[5] Christian Decker. *On the scalability and security of bitcoin*. PhD thesis, ETH Zurich, 2016.

[6] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. eltoo: A simple layer2 protocol for bitcoin. *White paper: https://blockstream. com/eltoo. pdf*, 2018.

[7] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[8] Thaddeus Dryja. Discreet log contracts. https://adiabat.github.io/dlc.pdf, 2017. Accessed: 2020-02-26.

[9] Steve Ellis, Ari Juels, and Sergey Nazarov. Chainlink: A decentralized oracle network.(2017). https://link.smartcontract.com/whitepaper, 2017. Accessed: 2020-02-26.

[10] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*, page InPrint, 2020.

[11] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. Outpost: A responsive lightweight watchtower. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, AFT '19, page 31–40. Association for Computing Machinery, 2019.

[12] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf, 2009. Accessed: 2020-02-26.

[13] Lightning Network. Bolt #2: Peer protocol for channel management. https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md. Accessed: 2020-02-26.

[14] Lightning Network. Bolt #3: Peer protocol for channel management. https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md. Accessed: 2020-02-26.

[15] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf, 2016. Accessed: 2020-02-26.

[16] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

Appendix A. Transaction scripts

For completeness, this appendix lists the scripts for the transactions that are used in the establishment and update of DLC channels.

Script 1 (Fund transaction output script).

```
OP_2
<alice public key>
<bob   public key>
OP_2
OP_CHECKMULTISIG
```

Script 2 (Script for the first output of an update transaction).

```
OP_IF
    <delay 1>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    OP_2
        <alice public key>
        <bob   public key>
    OP_2
    OP_CHECKMULTISIG
OP_ELSE
    OP_IF
        <delay 2>
        OP_CHECKSEQUENCEVERIFY
        OP_DROP
        <alice/bob public key>
    OP_ELSE
        <alice/bob revocation public key>
    OP_ENDIF
    OP_CHECKSIG
OP_ENDIF
```

Note that `delay 1` must be less than `delay 2`.

Script 3 (Script for the second output of an update transaction).

```
OP_IF
    OP_2
        <alice public key>
        <bob   public key>
    OP_2
    OP_CHECKMULTISIG
OP_ELSE
    <delay 2>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <alice/bob public key>
    OP_CHECKSIG
OP_ENDIF
```

Note that `delay 2` should be the same value as used in the first output.

Script 4 (Buffer transaction output script).

```
OP_2
    <alice public key>
    <bob   public key>
OP_2
OP_CHECKMULTISIG
```

Script 5 (Script for the first output of a CET).

```
OP_IF
    <alice/bob public key>
      + <oracle message public key>
OP_ELSE
    <delay 3>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <bob/alice public key>
OP_ENDIF
OP_CHECKSIG
```

Note that the + operator represents elliptic curve point addition.

Script 6 (Output script of a split transaction).

```
OP_IF
    <alice/bob revocation public key>
OP_ELSE
    <delay>
    OP_CHECKSEQUENCEVERIFY
    OP_DROP
    <alice/bob public key>
OP_ENDIF
OP_CHECKSIG
```