Recursive Virtual Payment Channels for Bitcoin

Aggelos Kiayias
Chair in Cyber Security and Privacy
University of Edinburgh
School of Informatics
Edinburgh, United Kingdom
Chief scientist
IOHK
global
akiayias@inf.ed.ac.uk

Orfeas Stefanos Thyfronitis Litos
PhD candidate
University of Edinburgh
School of Informatics
Edinburgh, United Kingdom
o.thyfronitis@ed.ac.uk

ABSTRACT

Blockchains are slow. Layer-2 largely solves this problem. PCNs constitute the most prominent layer-2/off-chain protocols. LN is the most widely used PCN and works on Bitcoin. Opening a channel requires 1 on-chain transaction, which can at times be avoided by performing a multi-hop payment. Then however fees to the intermediaries must be paid, routing becomes an issue, payment delay is proportional to the number of intermediaries and per-payment privacy suffers.

We propose Recursive Channels, which allow for new channels to be opened on top of an arbitrarily long path of existing channels in a recursive manner (i.e. the preexisting channels may themselves be virtual), answering the question of feasibility in the affirmative.

Our construction relies on the proposed ANYPREVOUT signature type.

ACM Reference Format:
Aggelos Kiayias and Orfeas Stefanos Thyfronitis Litos. 2021. Recursive Virtual Payment Channels for Bitcoin. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The popularity of blockchains in recent years has stretched their performance to its limits. Due to their need for synchronisation their latency is large (e.g. Bitcoin has a latency of 1h [1]) and due to the need for massive redundancy their throughput is low (Bitcoin can handle at most 7 transactions per second [2]). To circumvent these inherent limitations of blockchains, a prominent solution is to optimistically handle payments off-chain via a Payment Channel Network (PCN) TODO: cite PCN SoK/many papers and only use the blockchain as an arbiter in case of dispute.

The most popular PCN is the Lightning Network (LN) [3], which works on top of Bitcoin. With this, parties can open a pairwise channel with a single on-chain transaction and subsequently pay each other an unlimited number of times, only limited by the speed of their internet connection. What is more, a party can pay another even if they do not have a direct channel. They can instead leverage a path of channels for a fee and perform a so-called multi-hop payment in an atomic manner. Unfortunately a multi-hop payment needs active cooperation by all intermediaries, therefore increasing the latency and the probability of failure of the payment.

To mitigate this issue, virtual payment channels have been proposed TODO: cite. These enable two parties, say Alice and Bob, to open a payment channel over two preexisting channels, one between Alice and Charlie and another between Charlie and Bob. TODO: check if recursive channels exist

However, due to the limited scripting language of Bitcoin, it has proved challenging to build a secure protocol that allows virtual channels to be opened over more than two underlying channels, TODO: delete following phrase if the previous's TODO answer is affirmative as well as to make this construction recursive in the sense that further virtual channels can be opened on top of other virtual channels.

This work fills this gap by providing a concrete protocol that allows for arbitrarily many channels to be opened on top of arbitrarily long channel paths, where the underlying channels may themselves be virtual. This is achieved using standard Bitcoin script and an elaborate transaction configuration. We formally prove the security of the protocol in the UC [4] setting. The construction relies on the ANYPREVOUT signature type, which does not sign the hash of the transaction it spends, therefore allowing for a single pre-signed transaction to spend any output with a suitable script. We conjecture that this primitive cannot be achieved without ANYPREVOUT.

2 RELATED WORK

Due to massive replication and pervasive synchronisation requirements, blockchains have inherently low throughput and high latency [2]. The most prominent solution is Payment Channel Networks (PCNs) [5]. A number of constructions has been proposed, each with its own features and limitations. One of the oldest PCN is Duplex Micropayment Channels (DMC) [6], which uses a decrementing timelock for each new channel state, ensuring that only the latest state can be put on-chain first. The Lightning Network [3] was proposed to allow channels to stay open for an arbitrary length of time. Lightning is now implemented and functional for Bitcoin. It has also been adapted for Ethereum [7], where it is known as the Raiden Network [8]. The wormhole attack [9] against Lightning

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

allowed colluding parties in a multi-hop payment to steal the fees of their intermediaries. Generalized Bitcoin-Compatible Channels [10] enable the creation of state channels on Bitcoin, extending channel functionality from simple payments to arbitrary Bitcoin scripts.

Payment routing [11, 12] is another research area that aims to improve the network efficiency without sacrificing privacy.

Sprites [13] leverages the scripting language of Ethereum to decrease the time collateral is locked up compared to Lightning. Perun [14] and GSCN [15] exploit the Turing-complete scripting language of Ethereum to provide virtual state channels, i.e. channels that can open without an on-chain transaction and that allow for arbitrary scripts to be executed off-chain. Similar features are provided by Celer [16]. Hydra [17] provides state channels for the Cardano [18] blockchain.

BDW [19] shows how pairwise channels over Bitcoin can be funded with no on-chain transactions by allowing parties to form groups that can pool their funds together off-chain and then use those funds to open channels. ACMU [20] allows for multi-path atomic payments with reduced collateral, enabling new applications such as crowdfunding conditional on reaching a funding target.

TEE-based [21] solutions [22–24] improve the throughput and efficiency of PCNs by an order of magnitude, at the cost of having to trust TEEs. Brick [25] uses a partially trusted committee to extend PCNs to fully asynchronous networks.

Solutions alternative to PCNs include sidechains [26] and partially centralised payment networks that entirely avoid using a blockchain [27–30].

Last but not least, a number of works propose virtual channel constructions for Bitcoin. Lightweight Virtual Payment Channels [31] enables a virtual channel to be opened on top of two preexisting channels and uses a technique similar to DMC. Bitcoin-Compatible Virtual Channels [32] also enables virtual channels on top of two preexisting simple (i.e. non-virtual) channels and offers two protocols, the first of which guarantees that the channel will stay off-chain for an agreed period, while the second allows the intermediary to turn the virtual into a simple channel.

[5], [11], [3], [8], [1], [2], [6], [27], [28], [29], [30], [15], [14], [22], [13], [12], [9], [26], [32], [10], [25], [20], [31], [17], [16], [23], [24]

3 HIGH LEVEL EXPLANATION

Conceptually, our protocol is split into three main actions: channel opening, payments and closing. A channel (P_1, P_n) between parties P_1 and P_n may be opened directly on-chain, in which case the two parties follow an opening procedure similar to that of LN, or it can be opened on top of a path of preexisting channels $(P_2, P_3), (P_3, P_4), \dots, (P_{n-3}, P_{n-2}), (P_{n-2}, P_{n-1})$. In the latter case all parties P_i on the path follow our novel protocol, setting aside funds in their channels as collateral for the new virtual channel that is being opened. Once all intermediaries are committed, P_1 and P_n finally create (and keep off-chain) their "commitment" transaction, following a logic similar to Lightning and thus their channel is open.

A payment over an established channel follows a procedure heavily inspired by LN, but without the use of HTLCs. To be completed, a payment needs three messages to be exchanged by the two parties.

Finally, the closing procedure of a channel C can be completed unilaterally and consists of signing and publishing a number of transactions on-chain. As we will discuss later, the exact transactions that a party will publish vary depending on the actions of the parties controlling the channels that form the "base" of C and the channels that are based on C. Our protocol can be augmented with a more efficient optimistic collaborative closing procedure, which however is left as future work.

In more detail, to open a channel (c.f. 15) the two counterparties (a.k.a. "endpoints") first create new keypairs and exchange the resulting public keys (2 messages), then prepare the underlying base channels if the new channel is virtual ($12 \cdot (n-1)$ total messages, i.e. 6 outgoing messages per endpoint and 12 outgoing messages per intermediary, for $n-2$ intermediaries), next they exchange signatures for their respective initial commitment transactions (2 messages) and lastly, if the channel is simple (i.e. not virtual), the "funder" signs and publishes the "funding" transaction on-chain. We here note that like LN, only one of the two parties, the funder, provides coins for a new channel. This limitation simplifies the execution model and the analysis, but can be lifted at the cost of additional protocol complexity.

3.1 Simple Channels

In a similar vein to earlier PCN proposals, having an open channel essentially means having very specific keys, transactions and signatures at hand, as well as checking the ledger periodically and being ready to take action if misbehaviour is detected. Let us first consider a simple channel that has been established between *Alice* and *Bob* where the former owns c_A and the latter c_B coins. There are three sets of transactions at play: A "funding" transaction that is put on-chain, off-chain "commitment" transactions that spend the funding output on channel closure and off-chain "revocation" transactions that spend commitment outputs in case of misbehaviour. TODO: add figure

In particular, there is a single on-chain funding transaction that spends $c_A + c_B$ funder's coins, with a single output that is encumbered with a $2/\{pk_{A,F}, pk_{B,F}\}$ multisig and carries $c_A + c_B$ coins.

Next, there are two commitment transactions, each of which can spend the funding tx and produce two outputs with c_A and c_B coins each. The two txs differ in the outputs' spending conditions: The c_A output in *Alice*'s commitment tx can be spent either by *Alice* after it has been on-chain for a pre-agreed period (i.e. it is encumbered with a "timelock"), or by a "revocation" transaction (discussed below) via a 2-of-2 multisig between the counterparties, whereas the c_B output can be spent only by *Bob* without a timelock. *Bob*'s commitment tx is symmetric: the c_A output can be spent only by *Alice* without timelock and the c_B output can be spent either by *Bob* after the timelock expiration or by a revocation tx. When a new pair of commitment txs are created (either during channel opening or on each update) *Alice* signs *Bob*'s commitment tx and sends him the signature (and vice-versa), therefore *Alice* can unilaterally sign and publish her commitment tx but not *Bob*'s (and vice-versa).

Last, there are $2m$ revocation transactions, where m is the total number of updates of the channel. The jth revocation tx held by an endpoint spends the output carrying the counterparty's funds in the counterparty's jth commitment tx. It has a single output spendable immediately by the aforementioned endpoint. Each endpoint stores m revocation txs, one for each superseded commitment tx. This creates a disincentive for an endpoint to cheat by using any other commitment transaction than its most recent one to close the channel: the timelock on the commitment output permits its counterparty to use the corresponding revocation transaction and thus claim the cheater's funds. Endpoints do not have a revocation tx for the last commitment transaction, therefore these can be safely published. For a channel update to be completed, the endpoints must exchange the signatures for the revocation txs that spend the commitment txs that just became obsolete.

Observe that the above logic is essentially a simplification of LN.

3.2 Virtual Channels

In order to gain intuition on how virtual channels function, consider $n-1$ simple channels established between n honest parties as before. P_1 (the funder) and P_n want to open a virtual channel over these base channels. Before opening the virtual, each base channel is entirely independent, having different unique keys, separate on-chain funding outputs, a possibly different balance and number of updates. After the n parties follow our novel virtual channel opening protocol, they will all hold off-chain a number of new, "virtual" transactions that spend their respective funding transactions and can themselves be spent by new commitment transactions in a manner that ensures fair funds allocation for all honest parties.

In particular, apart from the transactions of simple channels, each of the two endpoints also has an "initiator" transaction that spends the funding output of its only base channel and produces two outputs: one new funding output for the base channel and one "virtual" output (c.f. Figure TODO:). If the virtual transaction ends up on-chain, the latter output carries coins that will directly or indirectly fund the funding output of the virtual channel.

Intermediaries on the other hand store three sets of virtual transactions: "initiator", "extend-interval" and "merge-intervals" (c.f. Figure TODO:). Each intermediary has one initiator tx, which spends the party's two funding outputs and produces four: one funding output for each base channel, one output that directly pays the intermediary coins equal to the total value in the virtual channel, and one virtual output. If both funding outputs are still unspent, publishing its initiator tx is the only way for an intermediary to close either of its channels.

Furthermore, each intermediary has $O(n)$ extend-interval transactions. If exactly one of the party's two base channels' funding outputs is unspent, publishing an extend-interval transaction is the only way for the party to close that base channel. Such a transaction consumes two outputs: the only available funding output and a suitable virtual output, as discussed below. An extend-interval tx has three outputs: A funding output replacing the one just spent, one output that directly pays the intermediary coins equal to the total value of the virtual channel, and one virtual output.

Last, each intermediary has $O(n^2)$ merge-intervals transactions. If both party's base channels' funding outputs are spent, publishing a merge-intervals transaction is the only way for the party to close either base channel. Such a transaction consumes two suitable virtual outputs, as discussed below. It has two outputs: One that directly pays the intermediary coins equal to the total value of the virtual channel, and one virtual output.

To understand why this multitude of virtual transactions is needed, we now zoom out from the individual party and discuss the dynamic of the system as a whole. The first party P_i that wishes to close a base channel observes that its funding output(s) remain(s) unspent and publishes its initiator transaction. First, this allows P_i to use its commitment transaction to close the channel. Second, in case P_i is an intermediary, it directly regains the coins it has locked for the virtual channel. Third, it produces a virtual output that can only be consumed by P_{i-1} and P_{i+1}, the parties adjacent to P_i (if any) with specific extend-interval transactions. The virtual output of this extend-interval transaction can in turn be spent by specific extend-interval transactions of P_{i-2} or P_{i+2} that have not published a transaction yet (if any) and so on for the next neighbours. The idea is that each party only needs to publish a single virtual transaction to "collapse" the virtual layer and each virtual output uniquely defines the continuous interval of parties that have already published a virtual transaction and only allow parties at the edges of this interval to extend it. This prevents malicious parties from indefinitely replacing a virtual output with a new one. As the name suggests, merge-intervals transactions are published by parties that are adjacent to two parties that have already published their virtual transactions an in effect joins the two intervals into one.

Each virtual output can also be used as the funding output for the virtual channel after a timelock, to protect from unresponsive parties blocking the virtual channel indefinitely. This in turn means that if an intermediary observes either of its funding outputs being spent, it has to publish its suitable virtual transaction before the timelock expires to avoid losing funds. What is more, all virtual outputs need the signature of all parties to be spent before the timelock (i.e. they have an n-of-n multisig) in order to prevent colluding parties from faking the intervals progression. The only exception are virtual outputs that correspond to an interval that includes all parties, which can only be used as funding outputs for the virtual channel as its interval cannot be further extended, therefore the two separate spending methods and the associated timelock are dropped.

Many extend-interval and merge-intervals transactions have to be able to spend different outputs, depending on the order other base parties publish their virtual transactions. For example, P_3's extend-interval tx that extends the interval $\{P_1, P_2\}$ to $\{P_1, P_2, P_3\}$ must be able to spend both the virtual output of P_2's initiator transaction and P_2's extend-interval transaction which has spent P_1's initiator transaction (Figure TODO:). The same issue is faced by commitment transactions of a virtual channel, as any virtual output can potentially be used as the funding ouput for the channel. In order for the received signatures for virtual and commitment txs to be valid for multiple previous outputs, the previously proposed `ANYPREVOUT` sighash flag [33] is needed to be added to Bitcoin. We conjecture that recursive virtual channels cannot be constructed in Bitcoin without this flag. We hope this work provides additional motivation for this flag to be included in the future.

Note also that the newly established virtual channel can itself act as a base for further virtual channels, as its funding output can be unilaterally put on-chain in a pre-agreed maximum number of blocks. This in turn means that, as discussed later in more detail, a further virtual channel must take the delay of its virtual base channels into account to determine the timelocks needed for its own virtual outputs.

As for the actual protocol needed to establish a virtual channel, 6 chains of messages are exchanged, starting from the funder and hop by hop reaching the fundee and back (c.f. 11). The first communicates parties' identities, their funding keys and their neighbours' channel balances, the second creates new commitment transactions, the third circulates virtual keys, all parties' coins and desired timelocks, the fourth and the fifth circulate signatures for the virtual transactions (signatures for virtual outputs and funding outputs respectively) and the sixth circulates revocation signatures for the old channel states.

4 PRELIMINARIES & NOTATION

In this work we embrace the Universal Composition (UC) framework [4] to model parties, network interactions, adversarial influence and corruptions, as well as formalise and prove security.

UC closely follows and expands upon the simulation-based security paradigm [34]. For a particular real world protocol, the main goal of UC is allow us to provide a simple "interface", the ideal world functionality, that describes what the protocol achieves in an ideal way. The functionality takes the inputs of all protocol parties and knows which parties are corrupted, therefore it normally can achieve the intention of the protocol in a much more straightforward manner. At a high level, once we have the protocol and the functionality defined, our goal is to prove that no probabilistic polynomial-time (PPT) ITM can distinguish whether it is interacting with the real world protocol or the ideal world functionality. If this is true we then say that the protocol UC-realises the functionality.

The principal contribution of UC is the following: Once a functionality that corresponds to a particular protocol is found, any other higher level protocol that internally uses the former protocol can instead use the functionality. This allows cryptographic proofs to compose and obviates the need for re-proving the security of every underlying primitive in every new application that uses it, therefore vastly improving the efficiency and scalability of the effort of cryptographic proofs.

In UC, a number of interactive Turing Machines (ITMs) execute and send messages to each other. At each moment only one ITM is executing (has the "execution token") and when it sends a message to another ITM, it transfers the execution token to the receiver. Messages can be sent either locally (inputs, outputs) or over the network.

The first ITM to be activated is the environment $\mathcal{E}$. This can be any PPT ITM. This ITM encompasses everything that happens around the protocol under scrutiny, including the players that send instructions to the protocol. It also is the ITM that tries to distinguish whether it is in the real or the ideal world. Put otherwise, it plays the role of the distinguisher.

After activating and executing some code, $\mathcal{E}$ may input a message to any party. If this execution is in the real world, then each party is an ITM running the protocol Π. Otherwise if the execution takes place in the ideal world, then each party is a dummy that simply relays messages to the functionality $\mathcal{F}$. An activated real world party then follows its code, which may instruct it to parse its input and send a message to another party via the network.

In UC the network is fully controlled by the so called adversary $\mathcal{A}$, which may be any PPT ITM. Once activated by any network message, this machine can read the message contents and act adaptively, freely communicate with $\mathcal{E}$ bidirectionally, choose to deliver the message right away, delay its delivery arbitrarily long, even corrupt it or drop it entirely. Crucially, it can also choose to corrupt any protocol party (in other words, UC allows adaptive corruptions). Once a party is corrupted, its internal state, inputs, outputs and execution comes under the full control of $\mathcal{A}$ for the rest of the execution. Corruptions take place covertly, so other parties do not necessarily learn which parties are corrupt. Furthermore, a corrupted party cannot become honest again.

The fact that $\mathcal{A}$ controls the network in the real world is modelled by providing direct communication channels between $\mathcal{A}$ and every other machine. This however poses an issue for the ideal world, as $\mathcal{F}$ is a single party that replaces all real world parties, so the interface has to be adapted accordingly. Furthermore, if $\mathcal{F}$ is to be as simple as possible, simulating internally all real world parties is not the way forward. This however may prove necessary in order to faithfully simulate the messages that the adversary expects to see in the real world. To solve these issues an ideal world adversary, also known as simulator $\mathcal{S}$, is introduced. This party can communicate freely with $\mathcal{F}$ and completely engulfs the real world $\mathcal{A}$. It can therefore internally simulate real world parties and generate suitable messages so that $\mathcal{A}$ remains oblivious to the fact that this is the ideal world. Normally it just relays messages between $\mathcal{A}$ and $\mathcal{E}$.

From the point of view of the functionality, $\mathcal{S}$ is untrusted, therefore any information that $\mathcal{F}$ leaks to $\mathcal{S}$ has to be carefully monitored by the designer. Ideally it has to be as little as possible so that $\mathcal{S}$ does not learn more than what is needed to simulate the real world. This facilitates modelling privacy.

At any point during one of its activations, $\mathcal{E}$ may return a binary value. The entire execution then halts. Informally, we say that Π UC-realises $\mathcal{F}$, or equivalently that the ideal and the real worlds are indistinguishable, if $\forall$ PPT $\mathcal{A}$, $\exists$ PPT $\mathcal{S}$: $\forall$ PPT $\mathcal{E}$, the distance of the distributions over the machines' random tapes of the outputs of $\mathcal{E}$ in the two worlds is negligibly small. Note the order of quantifiers: $\mathcal{S}$ depends on $\mathcal{A}$, but not on $\mathcal{E}$.

5 MODEL & CONSTRUCTION

In this section we will examine the architecture and the details of our model, along with possible attacks and their mitigations. Following the UC framework [4], we define an ideal-world functionality $\mathcal{F}_{\text{Chan}}$ (Figures 1-5) and a simulator $\mathcal{S}$ (Figures 6-7), along with a real-world protocol Π_{Chan} (Figures 8-42) that UC-realizes $\mathcal{F}_{\text{Chan}}$ (Theorem 8.4).

Similarly to [35], the role of $\mathcal{E}$ corresponds to two distinct actors in a real world implementation. On the one hand $\mathcal{E}$ passes inputs that correspond to the desires of end-users (e.g. open a channel, pay, close), on the other hand $\mathcal{E}$ is responsible with periodically waking up parties to check the ledger and act upon any detected counterparty misbehaviour, similar to an always-on "daemon" that periodically nudges the implementation to perform these checks. Since it is possible that $\mathcal{E}$ fails to wake up a party often enough, Π_{Chan} explicitly checks whether it has become "negligent" every time it is activated and all security guarantees are conditioned on the party not being negligent.

Our ideal world functionality $\mathcal{F}_{\text{Chan}}$ represents a single channel, either simple or virtual. It acts as a relay between $\mathcal{A}$ and $\mathcal{E}$, leaking all messages. This simplifies the functionality and facilitates the indistinguishability argument by having $\mathcal{S}$ simply running internally the real world protocols of the channel parties Π_{Chan} with no modifications. $\mathcal{F}_{\text{Chan}}$ internally maintains a state machine (c.f. Figure TODO: state machine) that keeps track of which internal parties are corrupted or negligent, whether the channel has opened, whether a payment is underway, which external parties are to be considered trusted (as they correspond to other channels owned by the same player) and whether the channel has closed. The single security check performed is whether the on-chain coins are at least equal to the expected balance once the channel closes. If this check fails, $\mathcal{F}_{\text{Chan}}$ halts.

Our real world protocol Π_{Chan}, ran by party P, consists of two subprotocols: the Lightning-inspired part, dubbed LN (Figures 8-27) and the novel virtual layer subprotocol, named VIRT (Figures 28-42).

5.1 LN subprotocol

The LN subprotocol has two variations depending on whether P is the channel funder (*Alice*) or the fundee (*Bob*). It performs a number of tasks: Initialisation takes a single step for fundees and two steps for funders. LN first receives a public key $pk_{P,\text{out}}$ from $\mathcal{E}$. This is the public key that should eventually own all P's coins after the channel is closed. LN also initialises its internal variables. If P is a funder, LN waits for a second activation to generate a keypair and then waits for $\mathcal{E}$ to endow it with some coins, which will be subsequently used to open the channel (Figure 8).

After initialisation, the funder *Alice* is ready to open the channel. Once it is given by $\mathcal{E}$ *Bob*'s identity, the initial channel balance c and, in case it is a virtual, the identities of the base channel owners (Figure 15), *Alice* generates and sends *Bob* her funding and revocation public keys ($pk_{A,F}$, $pk_{A,R}$) along with c, $pk_{A,\text{out}}$, and the base channel identities (if any). Given that *Bob* has been initialised, it generates funding and revocation keys and replies to *Alice* with $pk_{B,F}$, $pk_{B,R}$, and $pk_{B,\text{out}}$ (Figure 10).

The next step prepares the base channels (Figure 11). If our channel is a simple one, then *Alice* simply generates the funding tx. If it is a virtual and assuming all base parties (running LN) cooperate, a chain of messages from *Alice* to *Bob* and back via all base parties is initiated (Figures 17 and 18). These messages let each successive neighbour know the identities of all the base parties. Furthermore each party instantiates a new "host" party that runs VIRT. It also generates new funding keys and communicates them, along with its out key and its leftward and rightward balances. If this circuit of messages completes, *Alice* delegates the creation of the new virtual layer transactions to its new VIRT host, which will be discussed later in detail. If the virtual layer is successful, each base party is informed by its host accordingly, intermediaries return to the OPEN state and *Alice* and *Bob* continue the opening procedure. In particular, *Alice* and *Bob* exchange signatures on the initial commitment transactions, therefore ensuring that the funding output can be spent (Figure 12). After that, in case the channel is simple the funding transaction is put on-chain (Figure 13) and finally $\mathcal{E}$ is informed of the successful channel opening.

There are two facts that should be noted: Firstly, in case the opened channel is virtual, each intermediary base party necessarily partakes in two channels. However each protocol instance only represents a party in a single channel, therefore each intermediary is in practice realised by two mutually trusted Π_{Chan} instances that communicate locally, called "siblings". Secondly, our protocol is not designed to gracefully recover if other parties do not send an expected message at any point in the opening or payment procedure. Such anti-Denial-of-Service measures would greatly complicate the protocol and are left as a task for a real world implementation. It should be however stressed that an honest party with an open channel that has fallen victim to such an attack can still unilaterally close the channel, therefore no coins are lost in any case.

Once the channel is open, *Alice* and *Bob* can carry out an unlimited number of payments in either direction with a speed that is bounded only by network delay. The payment procedure is identical for simple and virtual channels and crucially it does not implicate the intermediaries. For a payment to be carried out, the payee is first notified by $\mathcal{E}$ (Figure 22) and subsequently the payer is instructed by $\mathcal{E}$ to commence the payment (Figure 21).

If the channel is virtual, each party also checks that its upcoming balance is lower than the balance of its sibling's counterparty and that the upcoming balance of the counterparty is higher than the balance of its own sibling, otherwise it rejects the payment. This is to mitigate an attack where a malicious counterparty uses an old commitment transaction to spend the base funding output, therefore blocking the honest party from using its initiator virtual transaction. This check ensures that the coins gained by the punishment are sufficient to cover the losses from the blocked initiator transaction. If the attack takes place, other local channels based directly or indirectly on it are informed and they moved to a failed state. Note that this does not bring a risk of losing any of the total coins of all local channels. We conjecture that this balance constraint can be lifted if the current Lightning-based payment method is replaced with an eltoo-based one [36].

Subsequently each of the two parties builds the new commitment transaction of its counterparty, signs it and sends over the signature, then the revocation transactions for the previously valid commitment transactions are generated, signed and the signatures are exchanged. To reduce the number of messages, the payee sends the two signatures in one message. This does not put it at risk of losing funds, since the new commitment transaction (for which it has already received a signature and therefore can spend) gives it more funds than the previous one.

Π_{Chan} also monitors the chain for outdated commitment transactions by the counterparty and publishes the corresponding revocation transaction in case one is found (Figure 24). It also monitors whether the party is activated often enough and marks it as negligent otherwise (Figure 8). The need for explicit negligence marking stems from the fact that party activation is entirely controlled by $\mathcal{E}$, therefore it can happen that an otherwise honest party is not activated in time to prevent a malicious counterparty from successfully using an old commitment transaction. Therefore at the beginning of every activation while the channel is open, LN checks if the party has been activated within the last p blocks (where p is an implementation-dependent global constant TODO: decide if reference to Proposition is needed). If a party is marked as negligent, no balance security guarantees are given (c.f. Lemma 8.1). Note that this does not affect indistinguishability with the ideal world, as $\mathcal{F}_{\text{Chan}}$ is notified by our $\mathcal{S}$ if a party becomes negligent and does not perform the balance security check.

When either party is instructed by $\mathcal{E}$ to close the channel (Figure 26), it first asks its host to close (details on the exact steps are discussed later) and once that is done, the ledger is checked for any transaction spending the funding output. In case the latest remote commitment tx is on-chain, then the channel is already closed and no further action is necessary. If an old committment transaction is on-chain, the corresponding revocation transaction is used for punishment. If the funding output is still unspent, the party attempts to publish the latest commitment transaction after waiting for any relevant timelock to expire. Until the funding output is irrevocably spent, the party still has to periodically check the blockchain and again be ready to use a revocation transaction if an old commitment transaction spends the funding output after all (Figure 24).

5.2 VIRT subprotocol

This subprotocol acts as a mediator between the base channels and the Lightning-based logic. Put otherwise, its responsibility is putting on-chain the funding output of the channel when needed. When first initialised by a machine that executes the LN subprotocol (Figure 28), it learns and stores the identities, keys, and balances of various relevant parties, along with the required timelock and other useful data regarding the base channels. It then generates a number of keys as needed for the rest of the base preparation. If the initialiser is also the channel funder, then the VIRT machine initiates 4 "circuits" of messages. Each circuit consists of one message from the funder P_1 to its neighbour P_2, one message from each intermediary P_i to the "next" neighbour P_{i+1}, one message from the fundee P_n to its neighbour P_{n-1} and one more message from each intermediary P_i to the "previous" neighbour P_{i-1}, for a total of $2 \cdot (n-1)$ messages per circuit.

The first circuit (Figure 29) communicates all "out", virtual and funding keys (both old and new), all balances and all timelocks among all parties. In the second circuit (Figure 36) every party receives and verifies all signatures for all inputs of its virtual transactions that spend a virtual output. It also produces and sends its own such signatures to the other parties. Each party generates and circulates $S = \sum\limits_{i=2}^{n-2}(n-3+\chi_{i=2}+\chi_{i=n-1}+2(i-2+\chi_{i=2})(n-i-1+\chi_{i=n-1})) \in O(n^3)$ signatures (where χ_A is the characteristic function that equals 1 if A is true and 0 else), for a total of $nS \in O(n^4)$ signatures in this phase. On a related note, the number of virtual transactions stored by each party is 1 for the two endpoints (Figure 31) and $n-2+\chi_{i=2}+\chi_{i=n-1}+(i-2+\chi_{i=2})(n-i-1+\chi_{i=n-1}) \in O(n^2)$ for each intermediary (Figure 30). The third circuit concerns sharing signatures for the funding outputs (Figure 37). Each party signs all transactions that spend a funding output relevant to the party, i.e. the initiator transaction and some of the extend-interval transactions of its neighbours. The two endpoints send 2 signatures each when $n = 3$ and $n-2$ signatures each when $n > 3$, whereas each intermediary sends $2+\chi_{i+1<n}(n-2+\chi_{i=n-2})+\chi_{i-1>1}(n-2+\chi_{i=3}) \in O(n)$ signatures each. The last circuit of messages (Figure 38) carries the revocations of the previous states of all base channels. After this, base parties can only use the newly created virtual transactions to spend their funding outputs. In this step each party exchanges a single signature with each of its neighbours.

When VIRT is instructed to close (Figure 40), it first notifies its VIRT host (if any) and waits for it to close. After that, it signs and publishes the unique valid virtual transaction. It then repeatedly checks the chain to see if the transaction is included (Figure 41). If it is included, the virtual layer is closed and VIRT informs its higher layer. The instruction to close has to be received potentially many times, because a number of virtual transactions (the ones that spend the same output) are mutually exclusive and therefore if another base party publishes an incompatible virtual transaction contemporaneously and that remote transaction enters the chain, then our VIRT party has to try again with another, compatible virtual transaction.

6 SECURITY

The first step to formally arguing about the security of our scheme is to clearly delineate the exact security guarantees it provides. To that end, we first prove two similar claims regarding the conservation of funds in the real and ideal world, Lemmas 8.1 and 8.2 respectively. Informally, the first claims that an honest, non-negligent party which was implicated in an already closed channel on which a number of payments took place will have at least the expected funds on-chain. The second lemma states that for an ideal party in a similar situation, the balance that $\mathcal{F}_{\text{Chan}}$ has stored for it is at least equal to the expected funds. In both cases the expected funds are (initial balance - funds for supported virtuals - outbound payments + inbound payments). Note that the funds for supported virtuals only refer to those funds used by the funder of the virtual channel, not the rest of the base parties.

Both proofs follow the various possible execution paths, keeping track of the resulting balance in each case and coming to the conclusion that balance is secure in all cases, except if signatures are forged.

It is important to note that in fact Π_{Chan} provides a stronger guarantee, namely that an honest, non-negligent party with an open channel can unilaterally close it and obtain the expected funds on-chain within a known time frame, given that $\mathcal{E}$ sends the necessary "daemon" messages. This stronger guarantee is sufficient to make this construction reliable enough for real-world applications. However a corresponding ideal world functionality with such guarantees would have to be aware of the specific transactions and signatures, therefore it would be essentially as complicated as the protocol, thus violating the spirit of the simulation-based security paradigm.

Subsequently we prove Lemma 8.3, which informally states that if an ideal party and all its trusted parties are honest, then $\mathcal{F}_{\text{Chan}}$ does not halt with overwhelming probability. This is proven by first arguing that if the conditions of Lemma 8.2 for the ideal world hold, then the conditions of Lemma 8.1 also hold for the equivalent real world execution, therefore in this case $\mathcal{F}_{\text{Chan}}$ does not halt. We then argue that also in case the conditions of Lemma 8.2 do not hold, $\mathcal{F}_{\text{Chan}}$ may never halt as well, therefore concluding the proof.

We then formulate and prove Theorem 8.4, which states that Π_{Chan} UC-realises $\mathcal{F}_{\text{Chan}}$. The corresponding proof is a simple application of Lemma 8.3, the fact that $\mathcal{F}_{\text{Chan}}$ is a simple relay and that $\mathcal{S}$ faithfully simulates Π_{Chan} internally.

Lastly we construct a "multi-session extension" [37] of $\mathcal{F}_{\text{Chan}}$ and of Π_{Chan} and prove Theorem 8.6, which claims that the real-world multi-session extension protocol UC-realises the ideal-world multi-session extension functionality. The proof is straightforward and utilises the transitivity of UC-emulation.

All formal proofs can be found in the Appendix.

7 EVALUATION

8 FUTURE WORK

- Add support for cooperative adding multiple virtuals to single channel (needs cooperation by all hops of all existing virtuals of current channel)
- Add support for cooperative closing
- Use eltoo instead of lightning to avoid balance restriction that prevents the revoked-griefing attack
- Allow for user-defined "leeway" timeout and timeout renegotiation
- Incorporate fees
- Prevent DoS attacks
- Improve privacy (mention full $\mathcal{F}_{\text{Chan}}$ leakage)
- Integrate with Lightning (i.e. allow both HTLC multi-hop payments *and* virtual channels

Functionality $\mathcal{F}_{\text{Chan}}$ – general message handling rules

- On receiving (`msg`) by party R to $P \in \{Alice, Bob\}$ by means of `mode` $\in$ {input, output, network}, handle it according to the corresponding rule in Fig. 2, 3, 5, or 4 (if any) and subsequently send (RELAY, `msg`, P, $\mathcal{E}$, input) $\mathcal{A}$. // all messages are relayed to $\mathcal{A}$
- On receiving (RELAY, `msg`, P, R, `mode`) by $\mathcal{A}$ (`mode` $\in$ {input, output, network}, $P \in \{Alice, Bob\}$), relay `msg` to R as P by means of `mode`. // $\mathcal{A}$ fully controls outgoing messages by $\mathcal{F}_{\text{Chan}}$
- On receiving (INFO, `msg`) by $\mathcal{A}$, handle (`msg`) according to the corresponding rule in Fig. 2, 3, 5, or 4 (if any). After handling the message or after an "ensure" fails, send (HANDLED, `msg`) to $\mathcal{A}$. // (INFO, `msg`) messages by $\mathcal{S}$ always return control to $\mathcal{S}$ without any side-effect to any other ITI, except if $\mathcal{F}_{\text{Chan}}$ halts
- $\mathcal{F}_{\text{Chan}}$ keeps track of two state machines, one for each of *Alice*, *Bob*. If there are more than one suitable rules for a particular message, or if a rule matches the message for both parties, then both rule versions are executed. // the two rules act on different state machines, so the order of execution does not matter

Figure 1

Functionality $\mathcal{F}_{\text{Chan}}$ – state machine up to OPEN for $P \in \{Alice, Bob\}$

On first activation: // before handing the message
 $pk_P \leftarrow \bot$; $\texttt{host}_P \leftarrow \bot$; $\texttt{enabler}_P \leftarrow \bot$; $\texttt{balance}_P \leftarrow 0$;
 $State_P \leftarrow$ UNINIT

On (BECAME CORRUPTED OR NEGLIGENT, P) by $\mathcal{A}$ or on output (ENABLER USED REVOCATION) by $\texttt{host}_P$ when in any state:
 $State_P \leftarrow$ IGNORED

On (INIT, pk) to P by $\mathcal{E}$ when $State_P$ = UNINIT:
 $pk_P \leftarrow pk$
 $State_P \leftarrow$ INIT

On (OPEN, x, $\mathcal{G}_{\text{Ledger}}, \dots$) to *Alice* by $\mathcal{E}$ when $State_A$ = INIT:
 store x
 $State_A \leftarrow$ TENTATIVE BASE OPEN

On (BASE OPEN) by $\mathcal{A}$ when $State_A$ = TENTATIVE BASE OPEN:
 $\texttt{balance}_A \leftarrow x$
 $State_A \leftarrow$ OPEN

On (BASE OPEN) by $\mathcal{A}$ when $State_B$ = INIT:
 $State_B \leftarrow$ OPEN

On (OPEN, x, $\texttt{hops} \neq \mathcal{G}_{\text{Ledger}}, \dots$) to *Alice* by $\mathcal{E}$ when $State_A$ = INIT:
 store x
 $\texttt{enabler}_A \leftarrow \texttt{hops[0].left}$
 add $\texttt{enabler}_A$ to *Alice*'s trusted parties
 $State_A \leftarrow$ PENDING VIRTUAL OPEN

On output (FUNDED, `host`, …) to *Alice* by $\texttt{enabler}_A$ when $State_A$ = PENDING VIRTUAL OPEN:
 $\texttt{host}_A \leftarrow \texttt{host[0].left}$
 $State_A \leftarrow$ TENTATIVE VIRTUAL OPEN

On output (FUNDED, `host`, …) to *Bob* by ITI $R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$ when $State_B$ = INIT:
 $\texttt{enabler}_B \leftarrow R$
 add $\texttt{enabler}_B$ to *Bob*'s trusted parties
 $\texttt{host}_B \leftarrow \texttt{host}$
 $State_B \leftarrow$ TENTATIVE VIRTUAL OPEN

On (VIRTUAL OPEN) by $\mathcal{A}$ when $State_P$ = TENTATIVE VIRTUAL OPEN:
 if $P = Alice$ **then** $\texttt{balance}_P \leftarrow x$
 $State_P \leftarrow$ OPEN

Figure 2

Functionality $\mathcal{F}_{\text{Chan}}$ – payments state machine for $P \in \{Alice, Bob\}$

1: On (PAY, x) by $\mathcal{E}$ when $State_P$ = OPEN: // P pays $\bar{P}$
2: store x
3: $State_P \leftarrow$ TENTATIVE PAY

4: On (PAY) by $\mathcal{A}$ when $State_P$ = TENTATIVE PAY: // P pays $\bar{P}$
5: $State_P \leftarrow$ (SYNC PAY, x)

6: On (GET PAID, y) by $\mathcal{E}$ when $State_P$ = OPEN: // $\bar{P}$ pays P
7: store y
8: $State_P \leftarrow$ TENTATIVE GET PAID

9: On (PAY) by $\mathcal{A}$ when $State_P$ = TENTATIVE GET PAID: // $\bar{P}$ pays P
10: $State_P \leftarrow$ (SYNC GET PAID, x)

11: When $State_P$ = (SYNC PAY, x):
12: **if** $State_{\bar{P}} \in$ {IGNORED, (SYNC GET PAID, x)} **then**
13: $\texttt{balance}_P \leftarrow \texttt{balance}_P - x$
14: // if $\bar{P}$ honest, this state transition happens simultaneously with l. 21
15: $State_P \leftarrow$ OPEN
16: **end if**

17: When $State_P$ = (SYNC GET PAID, x):
18: **if** $State_{\bar{P}} \in$ {IGNORED, (SYNC PAY, x)} **then**
19: $\texttt{balance}_P \leftarrow \texttt{balance}_P + x$
20: // if $\bar{P}$ honest, this state transition happens simultaneously with l. 15
21: $State_P \leftarrow$ OPEN
22: **end if**

Figure 3

Functionality $\mathcal{F}_{\text{Chan}}$ – fundings state machine for $P \in \{Alice, Bob\}$

1: On input (FUND ME, $x, \ldots$) by ITI $R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$ when $State_P$ = OPEN:
2: store x
3: add R to P's trusted parties
4: $State_P \leftarrow$ PENDING FUND

5: When $State_P$ = PENDING FUND:
6: **if** we intercept the command "define new VIRT ITI `host`" by $\mathcal{A}$, routed through P **then**
7: store `host`
8: $State_P \leftarrow$ TENTATIVE FUND
9: continue executing $\mathcal{A}$'s command
10: **end if**

11: On (FUND) by $\mathcal{A}$ when $State_P$ = TENTATIVE FUND:
12: $State_P \leftarrow$ SYNC FUND

13: When $State_P$ = OPEN:
14: **if** we intercept the command "define new VIRT ITI `host`" by $\mathcal{A}$, routed through P **then**
15: store `host`
16: $State_P \leftarrow$ TENTATIVE HELP FUND
17: continue executing $\mathcal{A}$'s command
18: **end if**
19: **if** we receive a RELAY message with `msg` = (INIT, ..., `fundee`) addressed from P by $\mathcal{A}$ **then**
20: add `fundee` to P's trusted parties
21: continue executing $\mathcal{A}$'s command
22: **end if**

23: On (FUND) by $\mathcal{A}$ when $State_P$ = TENTATIVE HELP FUND:
24: $State_P \leftarrow$ SYNC HELP FUND

25: When $State_P$ = SYNC FUND:
26: **if** $State_{\bar{P}} \in$ {IGNORED, SYNC HELP FUND} **then**
27: $\texttt{balance}_P \leftarrow \texttt{balance}_P - x$
28: $\texttt{host}_P \leftarrow \texttt{host}$
29: // if $\bar{P}$ honest, this state transition happens simultaneously with l. 36
30: $State_P \leftarrow$ OPEN
31: **end if**

32: When $State_P$ = SYNC HELP FUND:
33: **if** $State_{\bar{P}} \in$ {IGNORED, SYNC FUND} **then**
34: $\texttt{host}_P \leftarrow \texttt{host}$
35: // if $\bar{P}$ honest, this state transition happens simultaneously with l. 30
36: $State_P \leftarrow$ OPEN
37: **end if**

Figure 4

Functionality $\mathcal{F}_{\text{Chan}}$ – closure state machine for $P \in \{Alice, Bob\}$

On (CLOSE) by $\mathcal{E}$ when $State_P$ = OPEN:
    $State_P \leftarrow$ CLOSING

On input (BALANCE) to P by R where R is trusted by P:
    if $State_P \notin$ {UNINIT, INIT, PENDING VIRTUAL OPEN, TENTATIVE VIRTUAL OPEN, TENTATIVE BASE OPEN, IGNORED, CLOSED} **then**
        reply (MY BALANCE, $\texttt{balance}_P$, pk_P, $\texttt{balance}_{\bar{P}}$, $pk_{\bar{P}}$)
    else
        reply (MY BALANCE, 0, pk_P, 0, $pk_{\bar{P}}$)
    end if

On (CLOSE, P) by $\mathcal{A}$ when $State_P \notin$ {UNINIT, INIT, PENDING VIRTUAL OPEN, TENTATIVE VIRTUAL OPEN, TENTATIVE BASE OPEN, IGNORED}:
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign ouput to Σ
    `coins` $\leftarrow$ sum of values of outputs exclusively spendable or spent by pk_P in Σ
    `balance` $\leftarrow$ $\texttt{balance}_P$
    for all P's trusted parties R **do**
        input (BALANCE) to R as P and extract $\texttt{balance}_R$, pk_R from response
        `balance` $\leftarrow$ `balance` + $\texttt{balance}_R$
        `coins` $\leftarrow$ `coins` + sum of values of outputs exclusively spendable or spent by pk_R in Σ
    end for
    if `coins` $\geq$ `balance` **then**
        $State_P \leftarrow$ CLOSED
    else // balance security is broken
        halt
    end if

Figure 5

Simulator $\mathcal{S}$ – general message handling rules

- On receiving (RELAY, `in_msg`, P, R, `in_mode`) by $\mathcal{F}_{\text{Chan}}$ (`in_mode` $\in$ {input, output, network}, $P \in \{Alice, Bob\}$), handle (`in_msg`) with the simulated party P as if it was received from R by means of `in_mode`. In case simulated P does not exist yet, initialise it as an LN ITI. If there is a resulting message `out_msg` that is to be sent by simulated P to R' by means of `out_mode` $\in$ {input, output, network}, send (RELAY, `out_msg`, P, R', `out_mode`) to $\mathcal{F}_{\text{Chan}}$.
- On receiving by $\mathcal{F}_{\text{Chan}}$ a message to be sent by P to R via the network, carry on with this action (i.e. send this message via the internal $\mathcal{A}$).
- Relay any other incoming message to the internal $\mathcal{A}$ unmodified.
- On receiving a message (`msg`) by the internal $\mathcal{A}$, if it is addressed to one of the parties that correspond to $\mathcal{F}_{\text{Chan}}$, handle the message internally with the corresponding simulated party. Otherwise relay the message to its intended recipient unmodified. // Other recipients are $\mathcal{E}$, $\mathcal{G}_{\text{Ledger}}$ or parties unrelated to $\mathcal{F}_{\text{Chan}}$

Given that $\mathcal{F}_{\text{Chan}}$ relays all messages and that we simulate the real-world machines that correspond to $\mathcal{F}_{\text{Chan}}$, the simulation is perfectly indistinguishable from the real world.

Figure 6

Simulator $\mathcal{S}$ – notifications to $\mathcal{F}_{\text{Chan}}$

- "P" refers one of the parties that correspond to $\mathcal{F}_{\text{Chan}}$.
- When an action in this Figure interrupts an ITI simulation, continue simulating from the interruption location once action is over/$\mathcal{F}_{\text{Chan}}$ hands control back.

On (CORRUPT) by $\mathcal{A}$, addresed to P:
    // After executing this code and getting control back from $\mathcal{F}_{\text{Chan}}$ (which always happens, c.f. Fig. 1), deliver (CORRUPT) to simulated P (c.f. Fig. 6.
    send (INFO, BECAME CORRUPTED OR NEGLIGENT, P) to $\mathcal{F}_{\text{Chan}}$

When simulated P sets variable `negligent` to True (Fig. 8, l. 7/Fig. 9, l. 26):
    send (INFO, BECAME CORRUPTED OR NEGLIGENT, P) to $\mathcal{F}_{\text{Chan}}$

When simulated honest *Alice* receives (OPEN, x, hops, ...) by $\mathcal{E}$:
    store hops // will be used to inform $\mathcal{F}_{\text{Chan}}$ once the channel is open

When simulated honest *Bob* receives (OPEN, x, hops, ...) by *Alice*:
    if *Alice* is corrupted **then** store hops // if *Alice* is honest, we already have hops. If *Alice* became corrupted after receiving (OPEN, ...), overwrite hops

When the last of the honest simulated $\mathcal{F}_{\text{Chan}}$'s parties moves to the OPEN *State* for the first time (Fig. 12, l. 19/Fig. 14, l. 5/Fig. 15, l. 18):
    if hops = $\mathcal{G}_{\text{Ledger}}$ **then**
        send (INFO, BASE OPEN) to $\mathcal{F}_{\text{Chan}}$
    else
        send (INFO, VIRTUAL OPEN) to $\mathcal{F}_{\text{Chan}}$
    end if

When (both $\mathcal{F}_{\text{Chan}}$'s simulated parties are honest and complete sending and receiving a payment (Fig. 20, ll. 6 and 21 respectively), or (when only one party is honest and (completes either receiving or sending a payment)): // also send this message if both parties are honest when Fig. 20, l. 6 is executed by one party, but its counterparty is corrupted before executing Fig. 20, l. 21
    send (INFO, PAY) to $\mathcal{F}_{\text{Chan}}$

When honest P executes Fig. 17, l. 20 or (when honest P executes Fig. 17, l. 18 and $\bar{P}$ is corrupted): // in the first case if $\bar{P}$ is honest, it has already moved to the new host, (Fig 38, ll. 7, 23): lifting to next layer is done
    send (INFO, FUND) to $\mathcal{F}_{\text{Chan}}$

When one of the honest simulated $\mathcal{F}_{\text{Chan}}$'s parties P moves to the CLOSED state (Fig. 24, l. 8 or l. 11):
    send (INFO, CLOSE, P) to $\mathcal{F}_{\text{Chan}}$

Figure 7

Process LN – init

1: // When not specified, input comes from and output goes to $\mathcal{E}$.
2: // The ITI knows whether it is *Alice* (funder) or *Bob* (fundee). The activated party is P and the counterparty is $\bar{P}$.
3: On every activation, before handling the message:
4:     **if** `last_poll` $\neq \bot \wedge$ *State* $\neq$ CLOSED **then** // channel is open
5:         input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
6:         **if** `last_poll` $+ p < |\Sigma|$ **then** // p is a global parameter
7:             `negligent` $\leftarrow$ True
8:         **end if**
9:     **end if**

10: On (INIT, $pk_{P,\text{out}}$):
11:     ensure *State* $= \bot$
12:     *State* $\leftarrow$ INIT
13:     store $pk_{P,\text{out}}$
14:     $(c_A, c_B, \texttt{locked}_A, \texttt{locked}_B) \leftarrow (0, 0, 0, 0)$
15:     $(\texttt{paid\_out}, \texttt{paid\_in}) \leftarrow (\emptyset, \emptyset)$
16:     `negligent` $\leftarrow$ False
17:     `last_poll` $\leftarrow \bot$
18:     output (INIT OK)

19: On (TOP UP):
20:     ensure P = *Alice* // activated party is the funder
21:     ensure *State* = INIT
22:     $(sk_{P,\text{chain}}, pk_{P,\text{chain}}) \leftarrow$ KEYGEN()
23:     input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
24:     output (TOP UP TO, $pk_{P,\text{chain}}$)
25:     **while** $\nexists$tx $\in \Sigma, c_{P,\text{chain}} : (c_{P,\text{chain}}, pk_{P,\text{chain}}) \in$ tx.outputs **do**
26:         // while waiting, all other messages by P are ignored
27:         wait for input (CHECK TOP UP)
28:         input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
29:     **end while**
30:     *State* $\leftarrow$ TOPPED UP
31:     output (TOP UP OK, $c_{P,\text{chain}}$)

32: On (BALANCE):
33:     ensure $State^P \in$ {OPEN, CLOSED}
34:     output (BALANCE, c_A, $pk_{A,\text{out}}$, c_B, $pk_{B,\text{out}}$, $\texttt{locked}_A$, $\texttt{locked}_B$)

Figure 8

TODO: use center and captionof instead of figure, like VIRT

Process LN – methods used by VIRT

REVOKEPREVIOUS():

  ensure *State* ∈ WAITING FOR (OUTBOUND) REVOCATION

  $R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.P, output: ($C_{P,i}$.outputs.P.value, $pk_{\bar{P},\text{out}}$)}

  $\text{sig}_{A,R,i} \leftarrow \text{SIGN}(R_{\bar{P},i}, sk_{P,R})$

  if *State* = WAITING FOR REVOCATION **then**

    State ← WAITING FOR INBOUND REVOCATION

  else // *State* = WAITING FOR OUTBOUND REVOCATION

    $i \leftarrow i + 1$

    State ← WAITING FOR HOSTS READY

  end if

  $\texttt{host}_P \leftarrow \texttt{host}'_P$ // forget old host, use new host instead

  `layer` ← `layer` + 1

  return $\text{sig}_{P,R,i}$

PROCESSREMOTEREVOCATION($\text{sig}_{\bar{P},R,i}$):

  ensure *State* = WAITING FOR (INBOUND) REVOCATION

  $R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: ($C_{\bar{P},i}$.outputs.$\bar{P}$.value, $pk_{P,\text{out}}$)}

  ensure $\text{VERIFY}(R_{P,i}, \text{sig}_{\bar{P},R,i}, pk_{\bar{P},R})$ = True

  if *State* = WAITING FOR REVOCATION **then**

    State ← WAITING FOR OUTBOUND REVOCATION

  else // *State* = WAITING FOR INBOUND REVOCATION

    $i \leftarrow i + 1$

    State ← WAITING FOR HOSTS READY

  end if

  return (OK)

NEGLIGENT():

  `negligent` ← True

  return (OK)

Figure 9

Process LN.EXCHANGEOPENKEYS()

$(sk_{A,F}, pk_{A,F}) \leftarrow$ KEYGEN(); $(sk_{A,R}, pk_{A,R}) \leftarrow$ KEYGEN()

State ← WAITING FOR OPENING KEYS

send (OPEN, c, `hops`, $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$) to `fundee`

// colored code is run by honest fundee. Validation is implicit

ensure we run the code of *Bob*

ensure *State* = INIT

store $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$

$(sk_{B,F}, pk_{B,F}) \leftarrow$ KEYGEN(); $(sk_{B,R}, pk_{B,R}) \leftarrow$ KEYGEN()

if `hops` = $\mathcal{G}_{\text{Ledger}}$ **then** // opening base channel

  `layer` ← 0

  $t_P \leftarrow s + p$ // s is the upper bound of η from Lemma 7.19 of [38]

  State ← WAITING FOR COMM SIG

else // opening virtual channel

  State ← WAITING FOR CHECK KEYS

end if

reply (ACCEPT CHANNEL, $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$)

ensure *State* = WAITING FOR OPENING KEYS

store $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$

State ← OPENING KEYS OK

Figure 10

Process LN.PREPAREBASE()

if `hops` = $\mathcal{G}_{\text{Ledger}}$ **then** // opening base channel

  $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 2/\{pk_{A,F}, pk_{B,F}\})$}

  $\texttt{host}_P \leftarrow \mathcal{G}_{\text{Ledger}}$

  `layer` ← 0

  $t_P \leftarrow s + p$

else // opening virtual channel

  input (FUND ME, *Alice*, *Bob*, `hops`, c, $pk_{A,F}$, $pk_{B,F}$) to `hops[0].left` and expect output (FUNDED, $\texttt{host}_P$, `funder_layer`, t_P) // ignore any other message

  `layer` ← `funder_layer`

end if

Figure 11

Process LN.EXCHANGEOPENSIGS()

// $s = (2 + \lceil \mathtt{maxTime}_{\mathtt{window}} + \frac{\mathtt{Delay}}{2} / \mathtt{minTime}_{\mathtt{window}} \rceil)\mathtt{windowSize}$, where $\mathtt{maxTime}_{\mathtt{window}}$, Delay, $\mathtt{minTime}_{\mathtt{window}}$ and windowSize are defined in Proposition ?? TODO: recheck and include proposition

$C_{A,0} \leftarrow$ TX {input: $(c, 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, (pk_{A,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}

$C_{B,0} \leftarrow$ TX {input: $(c, 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, pk_{A,\text{out}})$, $(0, (pk_{B,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}

$\text{sig}_{A,C,0} \leftarrow \text{SIGN}(C_{B,0}, sk_{A,F})$

State $\leftarrow$ WAITING FOR COMM SIG

send (FUNDING CREATED, $(c, pk_{A,\text{chain}})$, $\text{sig}_{A,C,0}$) to fundee

ensure *State* = WAITING FOR COMM SIG // if opening virtual channel, we have received (FUNDED, host_fundee) by hops[-1].right (Fig 14, l. 10)

if hops = $\mathcal{G}_{\text{Ledger}}$ **then** // opening base channel

  $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 2/\{pk_{A,F}, pk_{B,F}\})$}

end if

$C_{B,0} \leftarrow$ TX {input: $(c, 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, pk_{A,\text{out}})$, $(0, (pk_{B,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}

ensure $\text{VERIFY}(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F})$ = True

$C_{A,0} \leftarrow$ TX {input: $(c, 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, (pk_{A,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}

$\text{sig}_{B,C,0} \leftarrow \text{SIGN}(C_{A,0}, sk_{B,F})$

if hops = $\mathcal{G}_{\text{Ledger}}$ **then** // opening base channel

  State $\leftarrow$ WAITING TO CHECK FUNDING

else // opening virtual channel

  $c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$

  State $\leftarrow$ OPEN

end if

reply (FUNDING SIGNED, $\text{sig}_{B,C,0}$)

ensure *State* = WAITING FOR COMM SIG

ensure $\text{VERIFY}(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F})$ = True

Figure 12

Process LN.COMMITBASE()

$\text{sig}_F \leftarrow \text{SIGN}(F, sk_{A,\text{chain}})$

input (SUBMIT, (F, sig_F)) to $\mathcal{G}_{\text{Ledger}}$ // enter "while" below before sending

while $F \notin \Sigma$ **do**

  wait for input (CHECK FUNDING) // ignore all other messages

  input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ

end while

Figure 13

Process LN – external open messages for *Bob*

1: On input (CHECK FUNDING):

2:   ensure *State* = WAITING TO CHECK FUNDING

3:   input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ

4:   **if** $F \in \Sigma$ **then**

5:     *State* $\leftarrow$ OPEN

6:     reply (OPEN OK)

7:   **end if**

8: On output (FUNDED, host_P, funder_layer, t_P) by hops[-1].right:

9:   ensure *State* = WAITING FOR FUNDED

10:   store host_P // we will talk directly to host_P

11:   layer $\leftarrow$ funder_layer

12:   *State* $\leftarrow$ WAITING FOR COMM SIG

13:   reply (FUND ACK)

14: On output (CHECK KEYS, (pk_1, pk_2)) by hops[-1].right:

15:   ensure *State* = WAITING FOR CHECK KEYS

16:   ensure $pk_1 = pk_{A,F} \wedge pk_2 = pk_{B,F}$

17:   *State* $\leftarrow$ WAITING FOR FUDNED

18:   reply (KEYS OK)

Figure 14

Process LN – On (OPEN, c, hops, fundee):

// fundee is *Bob*

ensure we run the code of *Alice* // activated party is the funder

if hops = $\mathcal{G}_{\text{Ledger}}$ **then** // opening base channel

  ensure *State* = TOPPED UP

  ensure $c = c_{A,\text{chain}}$

else // opening virtual channel

  ensure len(hops) ≥ 2 // cannot open a virtual over 1 channel

end if

LN.EXCHANGEOPENKEYS()

LN.PREPAREBASE()

LN.EXCHANGEOPENSIGS()

if hops = $\mathcal{G}_{\text{Ledger}}$ **then**

  LN.COMMITBASE()

end if

input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ

last_poll $\leftarrow |\Sigma|$

$c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$

State $\leftarrow$ OPEN

output (OPEN OK, c, fundee, hops)

Figure 15

Process LN.UPDATEFORVIRTUAL()

$C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with $pk'_{P,F}$ and $pk'_{\bar{P},F}$ instead of $pk_{P,F}$ and $pk_{\bar{P},F}$ respectively, reducing the input and P's output by c_{virt}
$\text{sig}_{P,C,i+1} \leftarrow \text{SIGN}(C_{\bar{P},i+1})$ // kept by $\bar{P}$
send (UPDATE FORWARD, $\text{sig}_{P,C,i+1}$) to $\bar{P}$
// P refers to payer and $\bar{P}$ to payee both in local and remote code
$C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with $pk'_{P,F}$ and $pk'_{\bar{P},F}$ instead of $pk_{P,F}$ and $pk_{\bar{P},F}$ respectively, reducing the input and P's output by c_{virt}
ensure VERIFY($C_{\bar{P},i+1}$, $\text{sig}_{P,C,i+1}$, $pk'_{P,F}$) = True
$C_{P,i+1} \leftarrow C_{P,i}$ with $pk'_{\bar{P},F}$ and $pk'_{P,F}$ instead of $pk_{\bar{P},F}$ and $pk_{P,F}$ respectively, reducing the input and P's output by c_{virt}
$\text{sig}_{\bar{P},C,i+1} \leftarrow \text{SIGN}(C_{P,i+1}, sk'_{\bar{P},F})$ // kept by P
reply (UPDATE BACK, $\text{sig}_{\bar{P},C,i+1}$)
$C_{P,i+1} \leftarrow C_{P,i}$ with $pk'_{\bar{P},F}$ and $pk'_{P,F}$ instead of $pk_{\bar{P},F}$ and $pk_{P,F}$ respectively, reducing the input and P's output by c_{virt}
ensure VERIFY($C_{P,i+1}$, $\text{sig}_{\bar{P},C,i+1}$, $pk'_{\bar{P},F}$) = True

Figure 16

Process LN – virtualise start and end

On input (FUND ME, c_{virt}, `fundee`, `hops`, $pk_{A,V}$, $pk_{B,V}$) by `funder`:
  ensure *State* = OPEN
  ensure $c_P - \texttt{locked}_P \geq c_{\text{virt}}$
  State ← VIRTUALISING
  $(sk'_{P,F}, pk'_{P,F}) \leftarrow \text{KEYGEN}()$
  define new VIRT ITI $\texttt{host}'_P$
  send (VIRTUALISING, $\texttt{host}'_P$, $pk'_{P,F}$, `hops`, `fundee`, c_{virt}, 2, len(hops)) to $\bar{P}$ and expect reply (VIRTUALISING ACK, $\texttt{host}'_{\bar{P}}$, $pk'_{\bar{P},F}$)
  ensure $pk'_{\bar{P},F}$ is different from $pk_{\bar{P},F}$ and all older $\bar{P}$'s funding public keys
  LN.UPDATEFORVIRTUAL()
  State ← WAITING FOR REVOCATION
  input (HOST ME, `funder`, `fundee`, $\texttt{host}'_{\bar{P}}$, $\texttt{host}_P$, c_P, $c_{\bar{P}}$, c_{virt}, $pk_{A,V}$, $pk_{B,V}$, $(sk'_{P,F}, pk'_{P,F})$, $(sk_{P,F}, pk_{P,F})$, $pk_{\bar{P},F}$, $pk'_{\bar{P},F}$, $pk_{P,\text{out}}$, len(hops)) to $\texttt{host}'_P$

On output (HOSTS READY, t_P) by $\texttt{host}_P$: // $\texttt{host}_P$ is the new host, renamed in Fig. 9, l. 12
  ensure *State* = WAITING FOR HOSTS READY
  State ← OPEN
  move $pk_{P,F}$, $pk_{\bar{P},F}$ to list of old funding keys
  $(sk_{P,F}, pk_{P,F}) \leftarrow (sk'_{P,F}, pk'_{P,F})$; $pk_{\bar{P},F} \leftarrow pk'_{\bar{P},F}$
  if len(hops) = 1 **then** // we are the last hop
    output (FUNDED, $\texttt{host}_P$, `layer`, t_P) to `fundee` and expect reply (FUND ACK)
  else if we have received input FUND ME just before we moved to the VIRTUALISING state **then** // we are the first hop
    $c_P \leftarrow c_P - c_{\text{virt}}$
    output (FUNDED, $\texttt{host}_P$, `layer`, t_P) to `funder` // do not expect reply by `funder`
  end if
  reply (HOST ACK)

Figure 17

Process LN – virtualise hops

On (VIRTUALISING, $\text{host}'_{\bar{P}}$, $pk'_{\bar{P},F}$, hops, fundee, c_{virt}, i, n) by $\bar{P}$:
    ensure *State* = OPEN
    ensure $c_{\bar{P}} - \text{locked}_{\bar{P}} \geq c$; $1 \leq i \leq n$
    ensure $pk'_{\bar{P},F}$ is different from $pk_{\bar{P},F}$ and all older $\bar{P}$'s funding public keys
    State ← VIRTUALISING
    $\text{locked}_{\bar{P}} \leftarrow \text{locked}_{\bar{P}} + c$ // if $\bar{P}$ is hosting the funder, $\bar{P}$ will transfer c_{virt} coins instead of locking them, but the end result is the same
    $(sk'_{P,F}, pk'_{P,F}) \leftarrow$ KEYGEN()
    if len(hops) > 1 **then** // we are not the last hop
        define new VIRT ITI host'_P
        input (VIRTUALISING, host'_P, $(sk'_{P,F}, pk'_{P,F})$, $pk'_{\bar{P},F}$, $pk_{P,\text{out}}$, hops[1:], fundee, c_{virt}, $c_{\bar{P}}$, c_P, i, n) to hops[1].left and expect reply (VIRTUALISING ACK, host_sibling, $pk_{\text{sib},\bar{P},F}$)
        input (INIT, host_P, $\text{host}'_{\bar{P}}$, host_sibling, $(sk'_{P,F}, pk'_{P,F})$, $pk'_{\bar{P},F}$, $pk_{\text{sib},\bar{P},F}$, $(sk_{P,F}, pk_{P,F})$, $pk_{\bar{P},F}$, $pk_{P,\text{out}}$, c_P, $c_{\bar{P}}$, c_{virt}, i, t_P, "left", n) to host'_P and expect reply (HOST INIT OK)
    else // we are the last hop
        input (INIT, host_P, $\text{host}'_{\bar{P}}$, fundee=fundee, $(sk'_{P,F}, pk'_{P,F})$, $pk'_{\bar{P},F}$, $(sk_{P,F}, pk_{P,F})$, $pk_{\bar{P},F}$, $pk_{P,\text{out}}$, c_P, $c_{\bar{P}}$, c_{virt}, t_P, i, "left", n) to new VIRT ITI host'_P and expect reply (HOST INIT OK)
    end if
    State ← WAITING FOR REVOCATION
    send (VIRTUALISING ACK, host'_P, $pk'_{P,F}$) to $\bar{P}$

On input (VIRTUALISING, host_sibling, $(sk'_{P,F}, pk'_{P,F})$, $pk_{\text{sib},\bar{P},F}$, $pk_{\text{sib,out}}$, hops, fundee, c_{virt}, $c_{\text{sib,rem}}$, c_{sib}, i, n) by sibling:
    ensure *State* = OPEN
    ensure $c_P - \text{locked}_P \geq c$
    ensure $c_{\text{sib,rem}} \geq c_P \land c_{\bar{P}} \geq c_{\text{sib}}$ // avoid value loss by griefing attack: one counterparty closes with old version, the other stays idle forever
    State ← VIRTUALISING
    $\text{locked}_P \leftarrow \text{locked}_P + c$
    define new VIRT ITI host'_P
    send (VIRTUALISING, host'_P, $pk'_{P,F}$, hops, fundee, c_{virt}, $i+1$, n) to hops[0].right and expect reply (VIRTUALISING ACK, $\text{host}'_{\bar{P}}$, $pk'_{\bar{P},F}$)
    ensure $pk'_{\bar{P},F}$ is different from $pk_{\bar{P},F}$ and all older $\bar{P}$'s funding public keys
    LN.UPDATEFORVIRTUAL()
    input (INIT, host_P, $\text{host}'_{\bar{P}}$, host_sibling, $(sk'_{P,F}, pk'_{P,F})$, $pk'_{\bar{P},F}$, $pk_{\text{sib},\bar{P},F}$, $(sk_{P,F}, pk_{P,F})$, $pk_{\bar{P},F}$, $pk_{\text{sib,out}}$, c_P, $c_{\bar{P}}$, c_{virt}, i, "right", n) to host'_P and expect reply (HOST INIT OK)
    State ← WAITING FOR REVOCATION
    output (VIRTUALISING ACK, host'_P, $pk'_{\bar{P},F}$) to sibling

Figure 18

Process LN.SIGNATURESROUNDTRIP()

$C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with x coins moved from P's to $\bar{P}$'s output
$\text{sig}_{P,C,i+1} \leftarrow$ SIGN$(C_{\bar{P},i+1}, sk_{P,F})$ // kept by $\bar{P}$
State ← WAITING FOR COMMITMENT SIGNED
send (PAY, x, $\text{sig}_{P,C,i+1}$) to $\bar{P}$
// P refers to payer and $\bar{P}$ to payee both in local and remote code
ensure *State* = WAITING TO GET PAID $\land\ x = y$
$C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with x coins moved from P's to $\bar{P}$'s output
ensure VERIFY$(C_{\bar{P},i+1}, \text{sig}_{P,C,i+1}, pk_{P,F})$ = True
$C_{P,i+1} \leftarrow C_{P,i}$ with x coins moved from P's to $\bar{P}$'s output
$\text{sig}_{\bar{P},C,i+1} \leftarrow$ SIGN$(C_{P,i+1}, sk_{\bar{P},F})$ // kept by P
$R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: $(c_{\bar{P}}, pk_{P,\text{out}})$}
$\text{sig}_{\bar{P},R,i} \leftarrow$ SIGN$(R_{P,i}, sk_{\bar{P},R})$
State ← WAITING FOR PAY REVOCATION
reply (COMMITMENT SIGNED, $\text{sig}_{\bar{P},C,i+1}$, $\text{sig}_{\bar{P},R,i}$)
ensure *State* = WAITING FOR COMMITMENT SIGNED
$C_{P,i+1} \leftarrow C_{P,i}$ with x coins moved from P's to $\bar{P}$'s output

Figure 19

Process LN.REVOCATIONSTRIP()

ensure VERIFY($C_{P,i+1}$, $\mathrm{sig}_{\bar{P},C,i+1}$, $pk_{\bar{P},F}$) = True
$R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.$\bar{P}$, output: ($c_{\bar{P}}$, $pk_{P,\mathrm{out}}$)}
ensure VERIFY($R_{P,i}$, $\mathrm{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
$R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.P, output: (c_P, $pk_{\bar{P},\mathrm{out}}$)}
$\mathrm{sig}_{P,R,i} \leftarrow$ SIGN($R_{\bar{P},i}$, $sk_{P,R}$)
add x to `paid_out`
$c_P \leftarrow c_P - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x; i \leftarrow i + 1$
State $\leftarrow$ OPEN
if $\mathtt{host}_P \neq \mathcal{G}_{\mathrm{Ledger}}$ $\wedge$ we have a `host_sibling` **then** // we are intermediary channel
  input (NEW BALANCE, c_P, $c_{\bar{P}}$) to $\mathtt{host}_P$
  relay message as input to `sibling` // run by VIRT
  relay message as output to `guest` // run by VIRT
  store new sibling balance and reply (NEW BALANCE OK)
  output (NEW BALANCE OK) to `sibling` // run by VIRT
  output (NEW BALANCE OK) to `guest` // run by VIRT
end if
send (REVOKE AND ACK, $\mathrm{sig}_{P,R,i}$) to $\bar{P}$
ensure *State* = WAITING FOR PAY REVOCATION
$R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: (c_P, $pk_{\bar{P},\mathrm{out}}$)}
ensure VERIFY($R_{\bar{P},i}$, $\mathrm{sig}_{P,R,i}$, $pk_{P,R}$) = True
add x to `paid_in`
$c_P \leftarrow c_P - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x; i \leftarrow i + 1$
State $\leftarrow$ OPEN
if $\mathtt{host}_P \neq \mathcal{G}_{\mathrm{Ledger}}$ $\wedge$ $\bar{P}$ has a `host_sibling` **then** // we are intermediary channel
  input (NEW BALANCE, $c_{\bar{P}}$, c_P) to $\mathtt{host}_{\bar{P}}$
  relay message as input to `sibling` // run by VIRT
  relay message as output to `guest` // run by VIRT
  store new sibling balance and reply (NEW BALANCE OK)
  output (NEW BALANCE OK) to `sibling` // run by VIRT
  output (NEW BALANCE OK) to `guest` // run by VIRT
end if

Figure 20

Process LN – On (PAY, x):

ensure *State* = OPEN $\wedge$ $c_P \geq x$
if $\mathtt{host}_P \neq \mathcal{G}_{\mathrm{Ledger}}$ $\wedge$ P has a `host_sibling` **then** // we are intermediary channel
  ensure $c_{\mathrm{sib,rem}} \geq c_P - x \wedge c_{\bar{P}} + x \geq c_{\mathrm{sib}}$ // avoid value loss by griefing attack: one counterparty closes with old version, the other stays idle forever
end if
LN.SIGNATURESROUNDTRIP()
LN.REVOCATIONSTRIP()
// No output is given to the caller, this is intentional

Figure 21

Process LN – On (GET PAID, y):

ensure *State* = OPEN $\wedge$ $c_{\bar{P}} \geq y$
if $\mathtt{host}_P \neq \mathcal{G}_{\mathrm{Ledger}}$ $\wedge$ P has a `host_sibling` **then** // we are intermediary channel
  ensure $c_P + y \leq c_{\mathrm{sib,rem}} \wedge c_{\mathrm{sib}} \leq c_{\bar{P}} - y$ // avoid value loss by griefing attack
end if
store y
State $\leftarrow$ WAITING TO GET PAID

Figure 22

Process LN – On (CHECK FOR LATERAL CLOSE):

if $\mathtt{host}_P \neq \mathcal{G}_{\mathrm{Ledger}}$ **then**
  input (CHECK FOR LATERAL CLOSE) to $\mathtt{host}_P$
end if

Figure 23

Process LN – On (CHECK CHAIN FOR CLOSED):

ensure *State* $\notin$ {$\perp$, INIT, TOPPED UP} // channel open
// even virtual channels check $\mathcal{G}_{\mathrm{Ledger}}$ directly. This is intentional
input (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ
`last_poll` $\leftarrow |\Sigma|$
if $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$ **then** // counterparty has closed maliciously
  State $\leftarrow$ CLOSING
  LN.SUBMITANDCHECKREVOCATION(j)
  State $\leftarrow$ CLOSED
  output (CLOSED)
else if $C_{P,i} \in \Sigma \vee C_{\bar{P},i} \in \Sigma$ **then**
  State $\leftarrow$ CLOSED
  output (CLOSED)
end if

Figure 24

Process LN – On output (ENABLER USED REVOCATION) by $\texttt{host}_P$:

$State \leftarrow$ BASE PUNISHED

Figure 27

Process LN.SUBMITANDCHECKREVOCATION(j)

$\text{sig}_{P,R,j} \leftarrow \text{SIGN}(R_{P,j}, sk_{P,R})$
input (SUBMIT, $(R_{P,j}, \text{sig}_{P,R,j}, \text{sig}_{\bar{P},R,j})$) to $\mathcal{G}_{\text{Ledger}}$
while $\nexists R_{P,j} \in \Sigma$ **do**
  wait for input (CHECK REVOCATION) // ignore other messages
  input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
end while
$c_P \leftarrow c_P + c_{\bar{P}}$
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then**
  input (USED REVOCATION) to $\texttt{host}_P$
end if

Figure 25

Process LN – On (CLOSE):

ensure $State \notin \{\perp$, INIT, TOPPED UP, CLOSED, BASE PUNISHED$\}$ // channel open
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then** // we have a virtual channel
  $State \leftarrow$ HOST CLOSING
  input (CLOSE) to $\texttt{host}_P$ and keep relaying any (CHECK IF CLOSING) or (CLOSE) input to $\texttt{host}_P$ until receiving output (CLOSED) by $\texttt{host}_P$
  $\texttt{host}_P \leftarrow \mathcal{G}_{\text{Ledger}}$
end if
$State \leftarrow$ CLOSING
input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
if $C_{\bar{P},i} \in \Sigma$ **then** // counterparty has closed honestly
  no-op // do nothing
else if $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$ **then** // counterparty has closed maliciously
  LN.SUBMITANDCHECKREVOCATION(j)
else // counterparty is idle
  while $\nexists$ unspent $\texttt{output} \in \Sigma$ that $C_{P,i}$ can spend **do** // possibly due to an active timelock
    wait for input (CHECK VIRTUAL) // ignore other messages
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
  end while
  $\text{sig}'_{P,C,i} \leftarrow \text{SIGN}(C_{P,i}, sk_{P,F})$
  input (SUBMIT, $(C_{P,i}, \text{sig}_{P,C,i}, \text{sig}'_{P,C,i})$) to $\mathcal{G}_{\text{Ledger}}$
end if

Figure 26

Process VIRT

On every activation, before handling the message:
  if $\texttt{last\_poll} \neq \perp$ **then** // virtual layer is ready
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    if $\texttt{last\_poll} + p < |\Sigma|$ **then**
      for $P \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$ **do** // at most 1 of funder, fundee is defined
        ensure P.NEGLIGENT() returns (OK)
      end for
    end if
  end if

// guest is trusted to give sane inputs, therefore a state machine and input verification are redundant
On input (INIT, $\texttt{host}_P$, $\bar{P}$, $\texttt{sibling}$, $\texttt{fundee}$, $(sk_{\text{loc,fund,new}}, pk_{\text{loc,fund,new}})$, $pk_{\text{rem,fund,new}}$, $pk_{\text{sib,rem,fund,new}}$, $(sk_{\text{loc,fund,old}}, pk_{\text{loc,fund,old}})$, $pk_{\text{rem,fund,old}}$, $pk_{\text{loc,out}}$, c_P, $c_{\bar{P}}$, c_{virt}, t_P, i, $\texttt{side}$, n) by $\texttt{guest}$:
  ensure $1 < i \leq n$ // host_funder ($i = 1$) is initialised with HOST ME
  ensure $\texttt{side} \in \{\text{“left”}, \text{“right”}\}$
  store message contents and $\texttt{guest}$ // sibling, $pk_{\text{sib},\bar{P},F}$ are missing for endpoints, fundee is present only in last node
  $(sk_{i,\text{fund,new}}, pk_{i,\text{fund,new}}) \leftarrow (sk_{\text{loc,fund,new}}, pk_{\text{loc,fund,new}})$
  $pk_{\text{myRem,fund,new}} \leftarrow pk_{\text{rem,fund,new}}$
  if $i < n$ **then** // we are not last hop
    $pk_{\texttt{sibRem},\text{fund,new}} \leftarrow pk_{\text{sib,rem,fund,new}}$
  end if
  if $\texttt{side} = \text{“left”}$ **then**
    $\texttt{side}' \leftarrow \text{“right”}$; $\texttt{myRem} \leftarrow i - 1$; $\texttt{sibRem} \leftarrow i + 1$
  else // $\texttt{side} = \text{“right”}$
    $\texttt{side}' \leftarrow \text{“left”}$; $\texttt{myRem} \leftarrow i + 1$; $\texttt{sibRem} \leftarrow i - 1$
  end if
  $(sk_{i,\texttt{side},\text{fund,old}}, pk_{i,\texttt{side},\text{fund,old}}) \leftarrow (sk_{\text{loc,fund,old}}, pk_{\text{loc,fund,old}})$
  $pk_{\text{myRem},\texttt{side}',\text{fund,old}} \leftarrow pk_{\text{rem,fund,old}}$
  if $\texttt{side} = \text{“left”}$ **then**
    $pk_{i,\text{out}} \leftarrow pk_{\text{loc,out}}$
  end if // otherwise sibling will send $pk_{i,\text{out}}$ in KEYS AND COINS FORWARD
  $(c_{i,\texttt{side}}, c_{\texttt{myRem},\texttt{side}'}, t_{i,\texttt{side}}) \leftarrow (c_P, c_{\bar{P}}, t_P)$
  $\texttt{last\_poll} \leftarrow \perp$
  if $\texttt{side} = \text{“left”} \wedge i \neq n$ **then**
    $(sk_{i,j,k}, pk_{i,j,k})_{j \in \{2,\ldots,n-1\}, k \in [n] \setminus \{j\}} \leftarrow \text{KEYGEN}()^{(n-2)(n-1)}$
  end if
  output (HOST INIT OK) to $\texttt{guest}$

On input (HOST ME, $\texttt{funder}$, $\texttt{fundee}$, $\bar{P}$, $\texttt{host}_P$, c_P, $c_{\bar{P}}$, c_{virt}, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $(sk_{1,\text{fund,new}}, pk_{1,\text{fund,new}})$, $(sk_{1,\text{right,fund,old}}, pk_{1,\text{right,fund,old}})$, $pk_{2,\text{left,fund,old}}$, $pk_{2,\text{left,fund,new}}$, $pk_{1,\text{out}}$, n) by $\texttt{guest}$:
  $\texttt{last\_poll} \leftarrow \perp$
  $i \leftarrow 1$
  $c_{1,\text{right}} \leftarrow c_P$; $c_{2,\text{left}} \leftarrow c_{\bar{P}}$
  $(sk_{1,j,k}, pk_{1,j,k})_{j \in \{2,\ldots,n-1\}, k \in [n] \setminus \{j\}} \leftarrow \text{KEYGEN}()^{(n-2)(n-1)}$
  ensure VIRT.CIRCULATEKEYSCOINSTIMES() returns (OK)
  ensure VIRT.CIRCULATEVIRTUALSIGS() returns (OK)
  ensure VIRT.CIRCULATEFUNDINGSIGS() returns (OK)
  ensure VIRT.CIRCULATEREVOCATIONS() returns (OK)

output (HOSTS READY, $p + \sum_{j=2}^{n-1}(s - 1 + t_j)$) to guest // p is every how many blocks we have to check the chain

Figure 28

Process VIRT.CIRCULATEKEYSCOINSTIMES(`left_data`):

if `left_data` is given as argument **then** // we are not `host_funder`

parse `left_data` as $((pk_{j,\text{fund,new}})_{j\in[i-1]}, (pk_{j,\text{left,fund,old}})_{j\in\{2,\dots,i-1\}}, (pk_{j,\text{right,fund,old}})_{j\in[i-1]}, (pk_{j,\text{out}})_{j\in[i-1]}, (c_{j,\text{left}})_{j\in\{2,\dots,i-1\}}, (c_{j,\text{right}})_{j\in[i-1]}, (t_j)_{j\in[i-1]}, pk_{\text{left,virt}}, pk_{\text{right,virt}}, (pk_{h,j,k})_{h\in[i-1],j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}})$

if we have a `sibling` **then** // we are not `host_fundee`

input (KEYS AND COINS FORWARD, (`left_data`, $(sk_{i,\text{left,fund,old}}, pk_{i,\text{left,fund,old}})$, $pk_{i,\text{out}}$, $c_{i,\text{left}}$, $t_{i,\text{left}}$, $(sk_{i,j,k}, pk_{i,j,k})_{j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}}$) to `sibling`

store input as `left_data` and parse it as $((pk_{j,\text{fund,new}})_{j\in[i-1]}, (pk_{j,\text{left,fund,old}})_{j\in\{2,\dots,i\}}, (pk_{j,\text{right,fund,old}})_{j\in[i-1]}, (pk_{j,\text{out}})_{j\in[i]}, (c_{j,\text{left}})_{j\in\{2,\dots,i\}}, (c_{j,\text{right}})_{j\in[i-1]}, (t_j)_{j\in[i-1]}, sk_{i,\text{left,fund,old}}, t_{i,\text{left}}, pk_{\text{left,virt}}, pk_{\text{right,virt}}, (pk_{h,j,k})_{h\in[i],j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}}, (sk_{i,j,k})_{j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}}$

$t_i \leftarrow \max(t_{i,\text{left}}, t_{i,\text{right}})$

replace $t_{i,\text{left}}$ in `left_data` with t_i

remove $sk_{i,\text{left,fund,old}}$ and $(sk_{i,j,k})_{j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}}$ from `left_data`

call VIRT.CIRCULATEKEYSCOINSTIMES(`left_data`) of $\bar{P}$ and assign returned value to `right_data`

parse `right_data` as $((pk_{j,\text{fund,new}})_{j\in\{i+1,\dots,n\}}, (pk_{j,\text{left,fund,old}})_{j\in\{i+1,\dots,n\}}, (pk_{j,\text{right,fund,old}})_{j\in\{i+1,\dots,n-1\}}, (pk_{j,\text{out}})_{j\in\{i+1,\dots,n\}}, (c_{j,\text{left}})_{j\in\{i+1,\dots,n\}}, (c_{j,\text{right}})_{j\in\{i+1,\dots,n-1\}}, (t_j)_{j\in\{i+1,\dots,n\}}, (pk_{h,j,k})_{h\in\{i+1,\dots,n\},j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}}$

output (KEYS AND COINS BACK, `right_data`, $(sk_{i,\text{right,fund,old}}, pk_{i,\text{right,fund,old}})$, $c_{i,\text{right}}$, t_i)

store output as `right_data` and parse it as $((pk_{j,\text{fund,new}})_{j\in\{i+1,\dots,n\}}, (pk_{j,\text{left,fund,old}})_{j\in\{i+1,\dots,n\}}, (pk_{j,\text{right,fund,old}})_{j\in\{i,\dots,n-1\}}, (pk_{j,\text{out}})_{j\in\{i+1,\dots,n\}}, (c_{j,\text{left}})_{j\in\{i+1,\dots,n\}}, (c_{j,\text{right}})_{j\in\{i,\dots,n-1\}}, (t_j)_{j\in\{i,\dots,n\}}, (pk_{h,j,k})_{h\in\{i+1,\dots,n\},j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}}, sk_{i,\text{right,fund,old}})$

remove $sk_{i,\text{right,fund,old}}$ from `right_data`

return (`right_data`, $pk_{i,\text{fund,new}}$, $pk_{i,\text{left,fund,old}}$, $pk_{i,\text{out}}$, $c_{i,\text{left}}$)

else // we are `host_fundee`

output (CHECK KEYS, $(pk_{\text{left,virt}}, pk_{\text{right,virt}})$) to `fundee` and expect reply (KEYS OK)

return ($pk_{n,\text{fund,new}}$, $pk_{n,\text{left,fund,old}}$, $pk_{n,\text{out}}$, $c_{n,\text{left}}$, t_n)

end if

else // we are `host_funder`

call VIRT.CIRCULATEKEYSCOINSTIMES($pk_{1,\text{fund,new}}$, $pk_{1,\text{right,fund,old}}$, $pk_{1,\text{out}}$, $c_{1,\text{right}}$, t_1, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $(pk_{1,j,k})_{j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}}$) of $\bar{P}$ and assign returned value to `right_data`

parse `right_data` as $((pk_{j,\text{fund,new}})_{j\in\{2,\dots,n\}}, (pk_{j,\text{left,fund,old}})_{j\in\{2,\dots,n\}}, (pk_{j,\text{right,fund,old}})_{j\in\{2,\dots,n-1\}}, (pk_{j,\text{out}})_{j\in\{2,\dots,n\}}, (c_{j,\text{left}})_{j\in\{2,\dots,n\}}, (c_{j,\text{right}})_{j\in\{2,\dots,n-1\}}, (t_j)_{j\in\{2,\dots,n\}}, (pk_{h,j,k})_{h\in\{2,\dots,n\},j\in\{2,\dots,n-1\},k\in[n]\setminus\{j\}})$

return (OK)

end if

Figure 29

Process VIRT

GETMIDTXS(i, n, c_{virt}, $c_{\text{rem,left}}$, $c_{\text{loc,left}}$, $c_{\text{loc,right}}$, $c_{\text{rem,right}}$, $pk_{\text{rem,left,fund,old}}$, $pk_{\text{loc,left,fund,old}}$, $pk_{\text{loc,right,fund,old}}$, $pk_{\text{rem,right,fund,old}}$, $pk_{\text{rem,left,fund,new}}$, $pk_{\text{loc,left,fund,new}}$, $pk_{\text{loc,right,fund,new}}$, $pk_{\text{rem,right,fund,new}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $pk_{\text{loc,out}}$, $(pk_{p,j,k})_{p\in[n],j\in[n-1]\setminus\{1\},k\in[n-1]\setminus\{1,j\}}$, $(pk_{p,2,1})_{p\in[n]}$, $(pk_{p,n-1,n})_{p\in[n]}$, $(t_j)_{j\in[n-1]\setminus\{1\}}$):

ensure $1 < i < n$

ensure $c_{\text{rem,left}} \geq c_{\text{virt}} \land c_{\text{loc,left}} \geq c_{\text{virt}}$ // left parties fund virtual channel

ensure $c_{\text{rem,left}} \geq c_{\text{loc,right}} \land c_{\text{rem,right}} \geq c_{\text{loc,left}}$ // avoid griefing attack

$c_{\text{left}} \leftarrow c_{\text{rem,left}} + c_{\text{loc,left}}$; $c_{\text{right}} \leftarrow c_{\text{loc,right}} + c_{\text{rem,right}}$

`left_old_fund` $\leftarrow 2/\{pk_{\text{rem,left,fund,old}}, pk_{\text{loc,left,fund,old}}\}$

`right_old_fund` $\leftarrow 2/\{pk_{\text{loc,right,fund,old}}, pk_{\text{rem,right,fund,old}}\}$

`left_new_fund` $\leftarrow 2/\{pk_{\text{rem,left,fund,new}}, pk_{\text{loc,left,fund,new}}\}$

`right_new_fund` $\leftarrow 2/\{pk_{\text{loc,right,fund,new}}, pk_{\text{rem,right,fund,new}}\}$

`virt_fund` $\leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}\}$

for all $j \in [n-1] \setminus \{1\}, k \in [n-1] \setminus \{1, j\}$ **do**

$all_{j,k} \leftarrow n/\{pk_{1,j,k}, \dots, pk_{n,j,k}\} \land$ "k"

end for

if $i = 2$ **then**

$all_{2,1} \leftarrow n/\{pk_{1,2,1}, \dots, pk_{n,2,1}\} \land$ "1"

end if

if $i = n - 1$ **then**

$all_{n-1,n} \leftarrow n/\{pk_{1,n-1,n}, \dots, pk_{n,n-1,n}\} \land$ "n"

end if

// After funding is complete, A_j has the signature of all other parties for all $all_{j,k}$ inputs, but other parties do not have A_j's signature for this input, therefore only A_j can publish it.

// $\text{TX}_{i,j,k}$:= i-th move, j, k input interval start and end. j, k unneeded for $i = 1$, k unneeded for $i = 2$.

$\text{TX}_1 \leftarrow$ TX:

inputs:

(c_{left}, `left_old_fund`),

(c_{right}, `right_old_fund`)

outputs:

($c_{\text{left}} - c_{\text{virt}}$, `left_new_fund`),

($c_{\text{right}} - c_{\text{virt}}$, `right_new_fund`),

(c_{virt}, $pk_{\text{loc,out}}$),

(c_{virt},

(if $(i - 1 > 1)$ then $all_{i-1,i}$ else False)

$\lor$ (if $(i + 1 < n)$ then $all_{i+1,i}$ else False)

$\lor$ (

if $(i - 1 = 1 \land i + 1 = n)$ then `virt_fund`

else if $(i - 1 > 1 \land i + 1 = n)$ then `virt_fund` $+ t_{i-1}$

else if $(i - 1 = 1 \land i + 1 < n)$ then `virt_fund` $+ t_{i+1}$

else /* $i-1>1 \land i+1<n$ */ $\texttt{virt\_fund} + \max(t_{i-1}, t_{i+1})$
)
)
if $i = 2$ **then**
$\mathrm{TX}_{2,1} \leftarrow \mathrm{TX}$:
inputs:
$(c_{\mathrm{virt}}, all_{2,1})$,
$(c_{\mathrm{right}}, \texttt{right\_old\_fund})$
outputs:
$(c_{\mathrm{right}} - c_{\mathrm{virt}}, \texttt{right\_new\_fund})$,
$(c_{\mathrm{virt}}, pk_{\mathrm{loc,out}})$,
$(c_{\mathrm{virt}}$,
if $(n > 3)$ then $(all_{3,2} \lor (\texttt{virt\_fund} + t_3))$
else $\texttt{virt\_fund}$
)
end if
if $i = n-1$ **then**
$\mathrm{TX}_{2,n} \leftarrow \mathrm{TX}$:
inputs:
$(c_{\mathrm{left}}, \texttt{left\_old\_fund})$,
$(c_{\mathrm{virt}}, all_{n-1,n})$
outputs:
$(c_{\mathrm{left}} - c_{\mathrm{virt}}, \texttt{left\_new\_fund})$,
$(c_{\mathrm{virt}}, pk_{\mathrm{loc,out}})$,
$(c_{\mathrm{virt}}$,
if $(n-2>1)$ then $(all_{n-2,n-1} \lor (\texttt{virt\_fund} + t_{n-2}))$
else $\texttt{virt\_fund}$
)
end if
for all $k \in \{2, \ldots, i-1\}$ **do** // $i-2$ txs
$\mathrm{TX}_{2,k} \leftarrow \mathrm{TX}$:
inputs:
$(c_{\mathrm{virt}}, all_{i,k})$,
$(c_{\mathrm{right}}, \texttt{right\_old\_fund})$
outputs:
$(c_{\mathrm{right}} - c_{\mathrm{virt}}, \texttt{right\_new\_fund})$,
$(c_{\mathrm{virt}}, pk_{\mathrm{loc,out}})$,
$(c_{\mathrm{virt}}$,
(if $(k-1>1)$ then $all_{k-1,i}$ else False)
$\lor$ (if $(i+1<n)$ then $all_{i+1,k}$ else False)
$\lor$ (
if $(k-1=1 \land i+1=n)$ then $\texttt{virt\_fund}$
else if $(k-1>1 \land i+1=n)$ then $\texttt{virt\_fund} + t_{k-1}$
else if $(k-1=1 \land i+1<n)$ then $\texttt{virt\_fund} + t_{i+1}$
else /* $k-1>1 \land i+1<n$ */ $\texttt{virt\_fund} + \max(t_{k-1}, t_{i+1})$
)
)
end for
for all $k \in \{i+1, \ldots, n-1\}$ **do** // $n-i-1$ txs
$\mathrm{TX}_{2,k} \leftarrow \mathrm{TX}$:
inputs:
$(c_{\mathrm{left}}, \texttt{left\_old\_fund})$
$(c_{\mathrm{virt}}, all_{i,k})$,
outputs:
$(c_{\mathrm{left}} - c_{\mathrm{virt}}, \texttt{left\_new\_fund})$,
$(c_{\mathrm{virt}}, pk_{\mathrm{loc,out}})$,
$(c_{\mathrm{virt}}$,
(if $(i-1>1)$ then $all_{i-1,k}$ else False)
$\lor$ (if $(k+1<n)$ then $all_{k+1,i}$ else False)
$\lor$ (
if $(i-1=1 \land k+1=n)$ then $\texttt{virt\_fund}$
else if $(i-1>1 \land k+1=n)$ then $\texttt{virt\_fund} + t_{i-1}$
else if $(i-1=1 \land k+1<n)$ then $\texttt{virt\_fund} + t_{k+1}$
else /* $i-1>1 \land k+1<n$ */ $\texttt{virt\_fund} + \max(t_{i-1}, t_{k+1})$
)
)
end for
if $i = 2$ **then** $m \leftarrow 1$ **else** $m \leftarrow 2$
if $i = n-1$ **then** $l \leftarrow n$ **else** $l \leftarrow n-1$
for all $(k_1, k_2) \in \{m, \ldots, i-1\} \times \{i+1, \ldots, l\}$ **do** // $(i-m)\cdot(l-i)$ txs
$\mathrm{TX}_{3,k_1,k_2} \leftarrow \mathrm{TX}$:
inputs:
$(c_{\mathrm{virt}}, all_{i,k_1})$,
$(c_{\mathrm{virt}}, all_{i,k_2})$
outputs:
$(c_{\mathrm{virt}}, pk_{\mathrm{loc,out}})$,
$(c_{\mathrm{virt}}$,
(if $(k_1-1>1)$ then $all_{k_1-1,\min(k_2,n-1)}$ else False)
$\lor$ (if $(k_2+1<n)$ then $all_{k_2+1,\max(k_1,2)}$ else False)
$\lor$ (
if $(k_1-1 \le 1 \land k_2+1 \ge n)$ then $\texttt{virt\_fund}$
else if $(k_1-1>1 \land k_2+1 \ge n)$ then $\texttt{virt\_fund} + t_{k_1-1}$
else if $(k_1-1 \le 1 \land k_2+1<n)$ then $\texttt{virt\_fund} + t_{k_2+1}$
else /* $k_1-1>1 \land k_2+1<n$ */
$\texttt{virt\_fund} + \max(t_{k_1-1}, t_{k_2+1})$
)
)
end for
return (
TX_1,
$(\mathrm{TX}_{2,k})_{k \in \{m,\ldots,l\} \setminus \{i\}}$,
$(\mathrm{TX}_{3,k_1,k_2})_{(k_1,k_2) \in \{m,\ldots,i-1\} \times \{i+1,\ldots,l\}}$
)

Figure 30

Process VIRT

// left and right refer to the two counterparties, with left being the one closer to the funder. Note difference with left/right meaning in VIRT.GETMIDTXS.
GETENDPOINTTX(i, n, c_{virt}, c_{left}, c_{right}, $pk_{\text{left,fund,old}}$, $pk_{\text{right,fund,old}}$, $pk_{\text{left,fund,new}}$, $pk_{\text{right,fund,new}}$ $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $(pk_{\text{all},j})_{j\in[n]}$, t):
  ensure $i \in \{1, n\}$
  ensure $c_{\text{left}} \geq c_{\text{virt}}$ // left party funds virtual channel
  $c_{\text{tot}} \leftarrow c_{\text{left}} + c_{\text{right}}$
  `old_fund` $\leftarrow 2/\{pk_{\text{left,fund,old}}, pk_{\text{right,fund,old}}\}$
  `new_fund` $\leftarrow 2/\{pk_{\text{left,fund,new}}, pk_{\text{right,fund,new}}\}$
  `virt_fund` $\leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}\}$
  if $i = 1$ **then** // funder's tx
    $all \leftarrow n/\{pk_{\text{all},1}, \dots, pk_{\text{all},n}\} \wedge$ "1"
  else // fundee's tx
    $all \leftarrow n/\{pk_{\text{all},1}, \dots, pk_{\text{all},n}\} \wedge$ "n"
  end if
  $\text{TX}_1 \leftarrow$ TX: // endpoints only have an "initiator" tx
    inputs:
      (c_{tot}, `old_fund`)
    outputs:
      ($c_{\text{tot}} - c_{\text{virt}}$, `new_fund`),
      (c_{virt}, $all \vee$ (`virt_fund` $+ t$))
  return TX_1

Figure 31

Process VIRT.SIBLINGSIGS()

parse input as $\text{sigs}_{\text{byLeft}}$
if $i = 2$ **then** $m \leftarrow 1$ **else** $m \leftarrow 2$
if $i = n - 1$ **then** $l \leftarrow n$ **else** $l \leftarrow n - 1$
$(\text{TX}_{i,1}$, $(\text{TX}_{i,2,k})_{k\in\{m,\dots,l\}\setminus\{i\}}$, $(\text{TX}_{i,3,k_1,k_2})_{(k_1,k_2)\in\{m,\dots,i-1\}\{i+1,\dots,l\}}) \leftarrow$ VIRT.GETMIDTXS(i, n, c_{virt}, $c_{i-1,\text{right}}$, $c_{i,\text{left}}$, $c_{i,\text{right}}$, $c_{i+1,\text{left}}$, $pk_{i-1,\text{right,fund,old}}$, $pk_{i,\text{left,fund,old}}$, $pk_{i,\text{right,fund,old}}$, $pk_{i+1,\text{left,fund,old}}$, $pk_{i-1,\text{fund,new}}$, $pk_{i,\text{fund,new}}$, $pk_{i,\text{fund,new}}$, $pk_{i+1,\text{fund,new}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $pk_{i,\text{out}}$, $(pk_{i,j,k})_{i\in[n],j\in[n-1]\setminus\{1\},k\in[n-1]\setminus\{1,j\}}$, $(pk_{i,2,1})_{i\in[n]}$, $(pk_{i,n-1,n})_{i\in[n]}$, $(t_i)_{i\in[n-1]\setminus\{1\}}$)

// notation: $\text{sig}(\text{TX}, pk)$:= sig with `ANYPREVOUT` flag such that VERIFY(TX, sig, pk) = True
ensure that the following signatures are present in $\text{sigs}_{\text{byLeft}}$ and store them:
- // $(l - m) \cdot (i - 1)$ signatures
  $\forall k \in \{m, \dots, l\} \setminus \{i\}, \forall j \in [i - 1]$:
    $\text{sig}(\text{TX}_{i,2,k}, pk_{j,i,k})$
- // $2 \cdot (i - m) \cdot (l - i) \cdot (i - 1)$ signatures
  $\forall k_1 \in \{m, \dots, i - 1\}, \forall k_2 \in \{i + 1, \dots, l\}, \forall j \in [i - 1]$:
    $\text{sig}(\text{TX}_{i,3,k_1,k_2}, pk_{j,i,k_1})$, $\text{sig}(\text{TX}_{i,3,k_1,k_2}, pk_{j,i,k_2})$
$\text{sigs}_{\text{toRight}} \leftarrow \text{sigs}_{\text{byLeft}}$

for all $j \in \{2, \dots, n - 1\} \setminus \{i\}$ **do**
  if $j = 2$ **then** $m' \leftarrow 1$ **else** $m' \leftarrow 2$
  if $j = n - 1$ **then** $l' \leftarrow n$ **else** $l' \leftarrow n - 1$
  $(\text{TX}_{j,1}$, $(\text{TX}_{j,2,k})_{k\in\{m',\dots,l'\}\setminus\{i\}}$, $(\text{TX}_{j,3,k_1,k_2})_{(k_1,k_2)\in\{m',\dots,i-1\}\{i+1,\dots,l'\}}) \leftarrow$ GETMIDTXS(j, n, c_{virt}, $c_{j-1,\text{right}}$, $c_{j,\text{left}}$, $c_{j,\text{right}}$, $c_{j+1,\text{left}}$, $pk_{j-1,\text{right,fund,old}}$, $pk_{j,\text{left,fund,old}}$, $pk_{j,\text{right,fund,old}}$, $pk_{j+1,\text{left,fund,old}}$, $pk_{j-1,\text{fund,new}}$, $pk_{j,\text{fund,new}}$, $pk_{j,\text{fund,new}}$, $pk_{j+1,\text{fund,new}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $pk_{j,\text{out}}$, $(pk_{k,p,s})_{k\in[n],p\in[n-1]\setminus\{1\},s\in[n-1]\setminus\{1,p\}}$, $(pk_{k,2,1})_{k\in[n]}$, $(pk_{k,n-1,n})_{k\in[n]}$, $(t_k)_{k\in[n-1]\setminus\{1\}}$)
  if $j < i$ **then** sigs $\leftarrow \text{sigs}_{\text{toLeft}}$ **else** sigs $\leftarrow \text{sigs}_{\text{toRight}}$
  for all $k \in \{m', \dots, l'\} \setminus \{j\}$ **do**
    add SIGN($\text{TX}_{j,2,k}$, $sk_{i,j,k}$, `ANYPREVOUT`) to sigs
  end for
  for all $k_1 \in \{m', \dots, j - 1\}, k_2 \in \{j + 1, \dots, l'\}$ **do**
    add SIGN($\text{TX}_{j,3,k_1,k_2}$, sk_{i,j,k_1}, `ANYPREVOUT`) to sigs
    add SIGN($\text{TX}_{j,3,k_1,k_2}$, sk_{i,j,k_2}, `ANYPREVOUT`) to sigs
  end for
end for
if $i + 1 = n$ **then** // next hop is `host_fundee`
  $\text{TX}_{n,1} \leftarrow$ VIRT.GETENDPOINTTX(n, n, c_{virt}, $c_{n-1,\text{right}}$, $c_{n,\text{left}}$, $pk_{n-1,\text{right,fund,old}}$, $pk_{n,\text{left,fund,old}}$, $pk_{n-1,\text{fund,new}}$, $pk_{n,\text{fund,new}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $(pk_{j,n-1,n})_{j\in[n]}$, t_{n-1})
end if

call $\bar{P}$.CIRCULATEVIRTUALSIGS($\text{sigs}_{\text{toRight}}$) and assign returned value to $\text{sigs}_{\text{byRight}}$
ensure that the following signatures are present in $\text{sigs}_{\text{byRight}}$ and store them:
- // $(l - m) \cdot (n - i)$ signatures
  $\forall k \in \{m, \dots, l\} \setminus \{i\}, \forall j \in \{i + 1, \dots, n\}$:
    $\text{sig}(\text{TX}_{i,2,k}, pk_{j,i,k})$
- // $2 \cdot (i - m) \cdot (l - i) \cdot (n - i)$ signatures
  $\forall k_1 \in \{m, \dots, i - 1\}, \forall k_2 \in \{i + 1, \dots, l\}, \forall j \in \{i + 1, \dots, n\}$:
    $\text{sig}(\text{TX}_{i,3,k_1,k_2}, pk_{j,i,k_1})$, $\text{sig}(\text{TX}_{i,3,k_1,k_2}, pk_{j,i,k_2})$
output (VIRTUALSIGSBACK, $\text{sigs}_{\text{toLeft}}$, $\text{sigs}_{\text{byRight}}$)

Figure 32

Process VIRT.INTERMEDIARYSIGS()

if $i = 2$ **then** $m \leftarrow 1$ **else** $m \leftarrow 2$
if $i = n - 1$ **then** $l \leftarrow n$ **else** $l \leftarrow n - 1$
$(\text{TX}_{i,1}$, $(\text{TX}_{i,2,k})_{k\in\{m,\dots,l\}\setminus\{i\}}$, $(\text{TX}_{i,3,k_1,k_2})_{(k_1,k_2)\in\{m,\dots,i-1\}\{i+1,\dots,l\}}) \leftarrow$ VIRT.GETMIDTXS(i, n, c_{virt}, $c_{i-1,\text{right}}$, $c_{i,\text{left}}$, $c_{i,\text{right}}$, $c_{i+1,\text{left}}$, $pk_{i-1,\text{right,fund,old}}$, $pk_{i,\text{left,fund,old}}$, $pk_{i,\text{right,fund,old}}$, $pk_{i+1,\text{left,fund,old}}$, $pk_{i-1,\text{fund,new}}$, $pk_{i,\text{fund,new}}$, $pk_{i,\text{fund,new}}$, $pk_{i+1,\text{fund,new}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $pk_{i,\text{out}}$, $(pk_{i,j,k})_{i\in[n],j\in[n-1]\setminus\{1\},k\in[n-1]\setminus\{1,j\}}$, $(pk_{i,2,1})_{i\in[n]}$, $(pk_{i,n-1,n})_{i\in[n]}$, $(t_i)_{i\in[n-1]\setminus\{1\}}$)
// not verifying our signatures in $\text{sigs}_{\text{byLeft}}$, our (trusted) `sibling` will do that
input (VIRTUAL SIGS FORWARD, $\text{sigs}_{\text{byLeft}}$) to `sibling`
VIRT.SIBLINGSIGS()
$\text{sigs}_{\text{toLeft}} \leftarrow \text{sigs}_{\text{byRight}} + \text{sigs}_{\text{toLeft}}$
if $i = 2$ **then** // previous hop is `host_funder`
  $\text{TX}_{1,1} \leftarrow$ VIRT.GETENDPOINTTX(1, n, c_{virt}, $c_{1,\text{right}}$, $c_{2,\text{left}}$, $pk_{1,\text{right,fund,old}}$, $pk_{2,\text{left,fund,old}}$, $pk_{1,\text{fund,new}}$, $pk_{2,\text{fund,new}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $(pk_{j,2,1})_{j\in[n]}$, t_2)

end if
return $\text{sigs}_{\text{toLeft}}$

Figure 33

Process $\textsc{virt.hostFundeeSigs}()$

$\text{TX}_{n,1} \leftarrow \textsc{virt.getEndpointTX}(n, n, c_{\text{virt}}, c_{n-1,\text{right}}, c_{n,\text{left}}, pk_{n-1,\text{right,fund,old}}, pk_{n,\text{right,fund,old}}, pk_{n-1,\text{fund,new}}, pk_{n,\text{fund,new}}, pk_{\text{left,virt}}, pk_{\text{right,virt}}, (pk_{j,n-1,n})_{j\in[n]}, t_{n-1})$
for all $j \in [n-1] \setminus \{1\}$ **do**
  if $j = 2$ **then** $m \leftarrow 1$ **else** $m \leftarrow 2$
  if $j = n-1$ **then** $l \leftarrow n$ **else** $l \leftarrow n-1$
  $(\text{TX}_{j,1}, (\text{TX}_{j,2,k})_{k\in\{m,\dots,l\}\setminus\{j\}}, (\text{TX}_{i,3,k_1,k_2})_{(k_1,k_2)\in\{m,\dots,i-1\}\{i+1,\dots,l\}}) \leftarrow \textsc{virt.getMidTXs}(j, n, c_{\text{virt}}, c_{j-1,\text{right}}, c_{j,\text{left}}, c_{j,\text{right}}, c_{j+1,\text{left}}, pk_{j-1,\text{right,fund,old}}, pk_{j,\text{left,fund,old}}, pk_{j,\text{right,fund,old}}, pk_{j+1,\text{left,fund,old}}, pk_{j-1,\text{fund,new}}, pk_{j,\text{fund,new}}, pk_{j,\text{fund,new}}, pk_{j+1,\text{fund,new}}, pk_{\text{left,virt}}, pk_{\text{right,virt}}, pk_{j,\text{out}}, (pk_{j,s,k})_{j\in[n],s\in[n-1]\setminus\{1\},k\in[n-1]\setminus\{1,s\}}, (pk_{j,2,1})_{j\in[n]}, (pk_{j,n-1,n})_{j\in[n]}, (t_j)_{j\in[n-1]\setminus\{1\}})$
  $\text{sigs}_{\text{toLeft}} \leftarrow \emptyset$
  for all $k \in \{m,\dots,l\} \setminus \{j\}$ **do**
    add $\textsc{sign}(\text{TX}_{j,2,k}, sk_{n,j,k}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toLeft}}$
  end for
  for all $k_1 \in \{m,\dots,j-1\}, k_2 \in \{j+1,\dots,l\}$ **do**
    add $\textsc{sign}(\text{TX}_{j,3,k_1,k_2}, sk_{n,j,k_1}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toLeft}}$
    add $\textsc{sign}(\text{TX}_{j,3,k_1,k_2}, sk_{n,j,k_2}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toLeft}}$
  end for
end for
return $\text{sigs}_{\text{toLeft}}$

Figure 34

Process $\textsc{virt.hostFunderSigs}()$

for all $j \in [n-1] \setminus \{1\}$ **do**
  if $j = 2$ **then** $m \leftarrow 1$ **else** $m \leftarrow 2$
  if $j = n-1$ **then** $l \leftarrow n$ **else** $l \leftarrow n-1$
  $(\text{TX}_{j,1}, (\text{TX}_{j,2,k})_{k\in\{m,\dots,l\}\setminus\{j\}}, (\text{TX}_{i,3,k_1,k_2})_{(k_1,k_2)\in\{m,\dots,i-1\}\{i+1,\dots,l\}}) \leftarrow \textsc{virt.getMidTXs}(j, n, c_{\text{virt}}, c_{j-1,\text{right}}, c_{j,\text{left}}, c_{j,\text{right}}, c_{j+1,\text{left}}, pk_{j-1,\text{right,fund,old}}, pk_{j,\text{left,fund,old}}, pk_{j,\text{right,fund,old}}, pk_{j+1,\text{left,fund,old}}, pk_{j-1,\text{fund,new}}, pk_{j,\text{fund,new}}, pk_{j,\text{fund,new}}, pk_{j+1,\text{fund,new}}, pk_{\text{left,virt}}, pk_{\text{right,virt}}, pk_{j,\text{out}}, (pk_{j,s,k})_{j\in[n],s\in[n-1]\setminus\{1\},k\in[n-1]\setminus\{1,s\}}, (pk_{j,2,1})_{j\in[n]}, (pk_{j,n-1,n})_{j\in[n]}, (t_j)_{j\in[n-1]\setminus\{1\}})$
  $\text{sigs}_{\text{toRight}} \leftarrow \emptyset$
  for all $k \in \{m,\dots,l\} \setminus \{j\}$ **do**
    add $\textsc{sign}(\text{TX}_{j,2,k}, sk_{1,j,k}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toRight}}$
  end for
  for all $k_1 \in \{m,\dots,j-1\}, k_2 \in \{j+1,\dots,l\}$ **do**
    add $\textsc{sign}(\text{TX}_{j,3,k_1,k_2}, sk_{1,j,k_1}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toRight}}$
    add $\textsc{sign}(\text{TX}_{j,3,k_1,k_2}, sk_{1,j,k_2}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toRight}}$
  end for
end for

call $\textsc{virt.circulateVirtualSigs}(\text{sigs}_{\text{toRight}})$ of $\bar{P}$ and assign output to $\text{sigs}_{\text{byRight}}$
$\text{TX}_{1,1} \leftarrow \textsc{virt.getEndpointTX}(1, n, c_{\text{virt}}, c_{1,\text{right}}, c_{2,\text{left}}, pk_{1,\text{right,fund,old}}, pk_{2,\text{left,fund,old}}, pk_{1,\text{fund,new}}, pk_{2,\text{fund,new}}, pk_{\text{left,virt}}, pk_{\text{right,virt}}, (pk_{j,2,1})_{j\in[n]}, t_2)$
return (OK)

Figure 35

Process $\textsc{virt.circulateVirtualSigs}(\text{sigs}_{\text{byLeft}})$

if $1 < i < n$ **then** // we are not host_funder nor host_fundee
  return $\textsc{virt.intermediarySigs}()$
else if $i = 1$ **then** // we are host_funder
  return $\textsc{virt.hostFunderSigs}()$
else if $i = n$ **then** // we are host_fundee
  return $\textsc{virt.hostFundeeSigs}()$
end if // it is always $1 \leq i \leq n$ – c.f. Fig. 28, l. 12 and l. 39

Figure 36

Process $\textsc{virt.circulateFundingSigs}(\text{sigs}_{\text{byLeft}})$

if $1 < i < n$ **then** // we are not endpoint
  if $i = 2$ **then** $m \leftarrow 1$ **else** $m \leftarrow 2$
  if $i = n-1$ **then** $l \leftarrow n$ **else** $l \leftarrow n-1$
  ensure that the following signatures are present in $\text{sigs}_{\text{byLeft}}$ and store them:
  - // 1 signature
    $\text{sig}(\text{TX}_{i,1}, pk_{i-1,\text{right,fund,old}})$
  - // $n - 3 + \chi_{i=2} + \chi_{i=n-1}$ signatures
    $\forall k \in \{m,\dots,l\} \setminus \{i\}$
      $\text{sig}(\text{TX}_{i,2,k}, pk_{i-1,\text{right,fund,old}})$
  input (VIRTUAL BASE SIG FORWARD, $\text{sigs}_{\text{byLeft}}$) to sibling
  extract and store $\text{sig}(\text{TX}_{i,1}, pk_{i-1,\text{right,fund,old}})$ and $\forall k \in \{m,\dots,l\} \setminus \{i\}$ $\text{sig}(\text{TX}_{i,2,k}, pk_{i-1,\text{right,fund,old}})$ from $\text{sigs}_{\text{byLeft}}$ // same signatures as sibling
  $\text{sigs}_{\text{toRight}} \leftarrow \{\textsc{sign}(\text{TX}_{i+1,1}, sk_{i,\text{right,fund,old}}, \text{ANYPREVOUT})\}$
  if $i + 1 < n$ **then**
    if $i + 1 = n - 1$ **then** $l' \leftarrow n$ **else** $l' \leftarrow n - 1$
    for all $k \in \{2,\dots,l'\}$ **do**
      add $\textsc{sign}(\text{TX}_{i+1,2,k}, sk_{i,\text{right,fund,old}}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toRight}}$
    end for
  end if
  call $\textsc{virt.circulateFundingSigs}(\text{sigs}_{\text{toRight}})$ of $\bar{P}$ and assign returned values to $\text{sigs}_{\text{byRight}}$
  ensure that the following signatures are present in $\text{sigs}_{\text{byRight}}$ and store them:
  - // 1 signature
    $\text{sig}(\text{TX}_{i,1}, pk_{i+1,\text{left,fund,old}})$
  - // $n - 3 + \chi_{i=2} + \chi_{i=n-1}$ signatures
    $\forall k \in \{m,\dots,l\} \setminus \{i\}$

$\mathrm{sig}(\mathrm{TX}_{i,2,k}, pk_{i+1,\mathrm{right,fund,old}})$
output (VIRTUAL BASE SIG BACK, $\mathrm{sigs}_{\mathrm{byRight}}$)
extract and store $\mathrm{sig}(\mathrm{TX}_{i,1}, pk_{i+1,\mathrm{right,fund,old}})$ and $\forall k \in \{m, \dots, l\} \setminus \{i\}$ $\mathrm{sig}(\mathrm{TX}_{i,2,k}, pk_{i+1,\mathrm{right,fund,old}})$ from $\mathrm{sigs}_{\mathrm{byRight}}$ // same signatures as sibling
$\mathrm{sigs}_{\mathrm{toLeft}} \leftarrow \{\mathrm{SIGN}(\mathrm{TX}_{i-1,1}, sk_{i,\mathrm{left,fund,old}}, \mathsf{ANYPREVOUT})\}$
if $i-1 > 1$ **then**
if $i-1=2$ **then** $m' \leftarrow 1$ **else** $m' \leftarrow 2$
for all $k \in \{m', \dots, n-1\}$ **do**
add $\mathrm{SIGN}(\mathrm{TX}_{i-1,2,k}, sk_{i,\mathrm{left,fund,old}}, \mathsf{ANYPREVOUT})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$
end for
end if
return $\mathrm{sigs}_{\mathrm{toLeft}}$
else if $i = 1$ **then** // we are host_funder
$\mathrm{sigs}_{\mathrm{toRight}} \leftarrow \{\mathrm{SIGN}(\mathrm{TX}_{2,1}, sk_{1,\mathrm{right,fund,old}}, \mathsf{ANYPREVOUT})\}$
if $2 = n-1$ **then** $l' \leftarrow n$ **else** $l' \leftarrow n-1$
for all $k \in \{3, \dots, l'\}$ **do**
add $\mathrm{SIGN}(\mathrm{TX}_{2,2,k}, sk_{1,\mathrm{right,fund,old}}, \mathsf{ANYPREVOUT})$ to $\mathrm{sigs}_{\mathrm{toRight}}$
end for
call VIRT.CIRCULATEFUNDINGSIGS($\mathrm{sigs}_{\mathrm{toRight}}$) of $\bar{P}$ and assign returned value to $\mathrm{sigs}_{\mathrm{byRight}}$
ensure that $\mathrm{sig}(\mathrm{TX}_{1,1}, pk_{2,\mathrm{left,fund,old}})$ is present in $\mathrm{sigs}_{\mathrm{byRight}}$ and store it
return (OK)
else if $i = n$ **then** // we are host_fundee
ensure $\mathrm{sig}(\mathrm{TX}_{n,1}, pk_{n-1,\mathrm{right,fund,old}})$ is present in $\mathrm{sigs}_{\mathrm{byLeft}}$ and store it
$\mathrm{sigs}_{\mathrm{toLeft}} \leftarrow \{\mathrm{SIGN}(\mathrm{TX}_{n-1,1}, sk_{n,\mathrm{left,fund,old}}, \mathsf{ANYPREVOUT})\}$
if $n-1=2$ **then** $m' \leftarrow 1$ **else** $m' \leftarrow 2$
for all $k \in \{m', \dots, n-2\}$ **do**
add $\mathrm{SIGN}(\mathrm{TX}_{n-1,2,k}, sk_{n,\mathrm{left,fund,old}}, \mathsf{ANYPREVOUT})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$
end for
return $\mathrm{sigs}_{\mathrm{toLeft}}$
end if // it is always $1 \le i \le n$ – c.f. Fig. 28, l. 12 and l. 39

Figure 37

Process VIRT.CIRCULATEREVOCATIONS(revoc_by_prev)

if revoc_by_prev is given as argument **then** // we are not host_funder
ensure guest.PROCESSREMOTEREVOCATION(revoc_by_prev) returns (OK)
else // we are host_funder
revoc_for_next $\leftarrow$ guest.REVOKEPREVIOUS()
input (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign ouput to Σ
last_poll $\leftarrow |\Sigma|$
call VIRT.CIRCULATEREVOCATIONS(revoc_for_next) of $\bar{P}$ and assign returned value to revoc_by_next
ensure guest.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK) // If the "ensure" fails, the opening process freezes, this is intentional. The channel can still close via (CLOSE)
return (OK)
end if
if we have a sibling **then** // we are not host_fundee nor host_funder
input (VIRTUAL REVOCATION FORWARD) to sibling
revoc_for_next $\leftarrow$ guest.REVOKEPREVIOUS()
input (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign ouput to Σ
last_poll $\leftarrow |\Sigma|$
call VIRT.CIRCULATEREVOCATIONS(revoc_for_next) of $\bar{P}$ and assign output to revoc_by_next
ensure guest.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK)
output (HOSTS READY, t_i) to guest and expect reply (HOST ACK)
output (VIRTUAL REVOCATION BACK)
end if
revoc_for_prev $\leftarrow$ guest.REVOKEPREVIOUS()
if $1 < i < n$ **then** // we are intermediary
output (HOSTS READY, t_i) to guest and expect reply (HOST ACK)
// p is every how many blocks we have to check the chain
else // we are host_fundee, case of host_funder covered earlier
output (HOSTS READY, $p + \sum_{j=2}^{n-1}(s-1+t_j)$) to guest and expect reply (HOST ACK)
end if
return revoc_for_prev

Figure 38

Process VIRT – poll

On input (CHECK FOR LATERAL CLOSE) by $R \in$ {guest, funder, fundee}:
input (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign output to Σ
$k_1 \leftarrow 0$
if $\mathrm{TX}_{i-1,1}$ is defined and $\mathrm{TX}_{i-1,1} \in \Sigma$ **then**
$k_1 \leftarrow i-1$
end if
for all $k \in [i-2]$ **do**
if $\mathrm{TX}_{i-1,2,k}$ is defined and $\mathrm{TX}_{i-1,2,k} \in \Sigma$ **then**
$k_1 \leftarrow k$
end if
end for
$k_2 \leftarrow 0$
if $\mathrm{TX}_{i+1,1}$ is defined and $\mathrm{TX}_{i+1,1} \in \Sigma$ **then**
$k_2 \leftarrow i+1$
end if
for all $k \in \{i+2, \dots, n\}$ **do**
if $\mathrm{TX}_{i+1,2,k}$ is defined and $\mathrm{TX}_{i+1,2,k} \in \Sigma$ **then**
$k_2 \leftarrow k$
end if
end for
last_poll $\leftarrow |\Sigma|$
if $k_1 > 0 \lor k_2 > 0$ **then** // at least one neighbour has published its TX
ignore all messages except for (CHECK IF CLOSING) by R
State $\leftarrow$ CLOSING
sigs $\leftarrow \emptyset$
end if
if $k_1 > 0 \land k_2 > 0$ **then** // both neighbours have published their TXs
add $(\mathrm{sig}(\mathrm{TX}_{i,3,k_1,k_2}, pk_{p,i,k_1}))_{p \in [n] \setminus \{i\}}$ to sigs
add $(\mathrm{sig}(\mathrm{TX}_{i,3,k_1,k_2}, pk_{p,i,k_2}))_{p \in [n] \setminus \{i\}}$ to sigs

add SIGN($\mathrm{TX}_{i,3,k_1,k_2}$, sk_{i,i,k_1}, ANYPREVOUT) to sigs
add SIGN($\mathrm{TX}_{i,3,k_1,k_2}$, sk_{i,i,k_2}, ANYPREVOUT) to sigs
input (SUBMIT, $\mathrm{TX}_{i,3,k_1,k_2}$, sigs) to $\mathcal{G}_{\text{Ledger}}$
else if $k_1 > 0$ **then** // only left neighbour has published its TX
add $(\text{sig}(\mathrm{TX}_{i,2,k_1}, pk_{p,i,k_1}))_{p \in [n] \setminus \{i\}}$ to sigs
add SIGN($\mathrm{TX}_{i,2,k_1}$, sk_{i,i,k_1}, ANYPREVOUT) to sigs
add SIGN($\mathrm{TX}_{i,2,k_1}$, $sk_{i,\text{left,fund,old}}$, ANYPREVOUT) to sigs
input (SUBMIT, $\mathrm{TX}_{i,2,k_1}$, sigs) to $\mathcal{G}_{\text{Ledger}}$
else if $k_2 > 0$ **then** // only right neighbour has published its TX
add $(\text{sig}(\mathrm{TX}_{i,2,k_2}, pk_{p,i,k_2}))_{p \in [n] \setminus \{i\}}$ to sigs
add SIGN($\mathrm{TX}_{i,2,k_2}$, sk_{i,i,k_2}, ANYPREVOUT) to sigs
add SIGN($\mathrm{TX}_{i,2,k_2}$, $sk_{i,\text{right,fund,old}}$, ANYPREVOUT) to sigs
input (SUBMIT, $\mathrm{TX}_{i,2,k_2}$, sigs) to $\mathcal{G}_{\text{Ledger}}$
end if

Figure 39

Process VIRT – On input (CLOSE) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$:

// At most one of funder, fundee is defined
if *State* = CLOSED **then** output (CLOSED) to R
if *State* = GUEST PUNISHED **then** output (GUEST PUNISHED) to R
ensure *State* $\in$ {OPEN, CLOSING}
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then** // $\texttt{host}_P$ is a VIRT
ignore all messages except for output (CLOSED) by $\texttt{host}_P$. Also relay to $\texttt{host}_P$ any (CHECK IF CLOSING) or (CLOSE) input received
input (CLOSE) to $\texttt{host}_P$
end if
// if we have a $\texttt{host}_P$, continue from here on output (CLOSED) by it
send (READ) to $\mathcal{G}_{\text{Ledger}}$ as R and assign reply to Σ
if $i \in \{1, n\} \land (\mathrm{TX}_{(i-1)+\frac{2}{n-1}(n-i),1} \in \Sigma \lor \exists k \in [n] : \mathrm{TX}_{(i-1)+\frac{2}{n-1}(n-i),2,k} \in \Sigma)$ **then** // we are an endpoint and our counterparty has closed – 1st subscript of TX is 2 if $i = 1$ and $n-1$ if $i = n$
ignore all messages except for (CHECK IF CLOSING) and (CLOSE) by R
State $\leftarrow$ CLOSING
give up execution token // control goes to $\mathcal{E}$
end if
let tx be the unique TX among $\mathrm{TX}_{i,1}$, $(\mathrm{TX}_{i,2,k})_{k \in [n]}$, $(\mathrm{TX}_{i,3,k_1,k_2})_{k_1,k_2 \in [n]}$ that can be appended to Σ in a valid way // ignore invalid subscript combinations
let sigs be the set of stored signatures that sign tx
add SIGN(tx, $sk_{i,\text{left,fund,old}}$, ANYPREVOUT), SIGN(tx, $sk_{i,\text{right,fund,old}}$, ANYPREVOUT), $(\text{SIGN}(\texttt{tx}, sk_{i,j,k}, \text{ANYPREVOUT}))_{j,k \in [n]}$ to sigs // ignore invalid signatures
ignore all messages except for (CHECK IF CLOSING) by R
State $\leftarrow$ CLOSING
send (SUBMIT, tx, sigs) to $\mathcal{G}_{\text{Ledger}}$

Figure 40

Process VIRT – On input (CHECK IF CLOSING) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$:

ensure *State* = CLOSING
send (READ) to $\mathcal{G}_{\text{Ledger}}$ as R and assign reply to Σ
if $i = 1$ **then** // we are host_funder
ensure that there exists an output with $c_P + c_{\bar{P}} - c_{\text{virt}}$ coins and a $2/\{pk_{1,\text{fund,new}}, pk_{2,\text{fund,new}}\}$ spending method with expired/non-existent timelock in Σ // new base funding output
ensure that there exists an output with c_{virt} coins and a $2/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}\}$ spending method with expired/non-existent timelock in Σ // virtual funding output
else if $i = n$ **then** // we are host_fundee
ensure that there exists an output with $c_P + c_{\bar{P}} - c_{\text{virt}}$ coins and a $2/\{pk_{n-1,\text{fund,new}}, pk_{n,\text{fund,new}}\}$ spending method with expired/non-existent timelock in Σ // new base funding output
ensure that there exists an output with c_{virt} coins and a $2/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}\}$ spending method with expired/non-existent timelock in Σ // virtual funding output
else // we are intermediary
if side = "left" **then** $j \leftarrow i - 1$ **else** $j \leftarrow i + 1$ // side is defined for all intermediaries – c.f. Fig. 28, l. 11
ensure that there exists an output with $c_P + c_{\bar{P}} - c_{\text{virt}}$ coins and a $2/\{pk_{i,\text{fund,new}}, pk_{j,\text{fund,new}}\}$ spending method with expired/non-existent timelock and an output with c_{virt} coins and a $pk_{i,\text{out}}$ spending method with expired/non-existent timelock in Σ
end if
State $\leftarrow$ CLOSED
output (CLOSED) to R

Figure 41

Process VIRT – punishment handling

On input (USED REVOCATION) by guest: // (USED REVOCATION) by funder/fundee is ignored
State $\leftarrow$ GUEST PUNISHED
input (USED REVOCATION) to $\texttt{host}_P$, expect reply (USED REVOCATION OK)
if funder or fundee is defined **then**
output (ENABLER USED REVOCATION) to it
else // sibling is defined
output (ENABLER USED REVOCATION) to sibling
end if

On input (ENABLER USED REVOCATION) by sibling:
State $\leftarrow$ GUEST PUNISHED
output (ENABLER USED REVOCATION) to guest

On output (USED REVOCATION) by $\texttt{host}_P$:
State $\leftarrow$ GUEST PUNISHED
if funder or fundee is defined **then**
output (ENABLER USED REVOCATION) to it
else // sibling is defined
output (ENABLER USED REVOCATION) to sibling
end if

Figure 42

LEMMA 8.1 (REAL WORLD BALANCE SECURITY). *Consider a real world execution with $P \in \{Alice, Bob\}$ honest LN ITI and $\bar{P}$ the counterparty ITI. Assume that all of the following are true:*

- *the internal variable* `negligent` *of P has value "False",*
- *P has transitioned to the* OPEN *State for the first time after having received* $(\text{OPEN}, c, \dots)$ *by either* $\mathcal{E}$ *or* $\bar{P}$*,*
- *P [has received* $(\text{FUND ME}, f_i, \dots)$ *as input by another* LN *ITI while State was* OPEN *and subsequently P transitioned to* OPEN *State] n times,*
- *P [has received* (PAY, d_i) *by* $\mathcal{E}$ *while State was* OPEN *and P subsequently transitioned to* OPEN *State] m times,*
- *P [has received* $(\text{GET PAID}, e_i)$ *by* $\mathcal{E}$ *while State was* OPEN *and P subsequently transitioned to* OPEN *State] l times.*

Let $\phi = 1$ *if* $P = Alice$*, or* $\phi = 0$ *if* $P = Bob$*. If P receives* (CLOSE) *by* $\mathcal{E}$ *and, if* $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ *the output of* $\texttt{host}_P$ *is* (CLOSED)*, then eventually the state obtained when P inputs* (READ) *to* $\mathcal{G}_{\text{Ledger}}$ *will contain h outputs each of value* c_i *and that has been spent or is exclusively spendable by* $pk_{R,\text{out}}$ *such that*

$$\sum_{i=1}^{h} c_i \geq \phi \cdot c - \sum_{i=1}^{n} f_i - \sum_{i=1}^{m} d_i + \sum_{i=1}^{l} e_i \tag{1}$$

with overwhelming probability in the security parameter, where R is a local, trusted machine (i.e. either P, P's `sibling`*, the party to which P sent* FUND ME *if such a message has been sent, or the* `sibling` *of one of the transitive closure of hosts of P).*

We first note that, as signature forgeries only happen with negligible probability and only a polynomial number of signatures are verified by honest parties throughout an execution, the event in which any forged signature passes the verification of an honest party or of $\mathcal{G}_{\text{Ledger}}$ happens only with negligible probability. We can therefore ignore this event throughout this proof and simply add a computationally negligible distance between $\mathcal{E}$'s outputs in the real and the ideal world at the end.

We also note that $pk_{P,\text{out}}$ has been provided by $\mathcal{E}$, therefore it can freely use coins spendable by this key. This is why we allow for any of the $pk_{P,\text{out}}$ outputs to have been spent.

Define the *history* of a channel as $H = (F, C)$, where each of F, C is a list of lists of integers. A party P which satisfies the Lemma conditions has a unique, unambiguously and recursively defined history: If the value hops in the $(\text{OPEN}, c, \text{hops}, \dots)$ message was equal to $\mathcal{G}_{\text{Ledger}}$, then F is the empty list, otherwise F is the concatenation of the F and C lists of the party that sent $(\text{FUNDED}, \dots)$ to P, as they were at the moment the latter message was sent. After initialised, F remains immutable. Observe that, if $\text{hops} \neq \mathcal{G}_{\text{Ledger}}$, both aforementioned messages must have been received before P transitions to the OPEN state.

The list C of party P is initialised to $[[g]]$ when P's *State* transitions for the first time to OPEN, where $g = c$ if $P = Alice$, or $g = 0$ if $P = Bob$; this represents the initial channel balance. The value x or $-x$ is appended to the last list in C when a payment is received (Fig. 20, l. 21) or sent (Fig. 20, l. 6) respectively by P. Moving on to the funding of new virtual channels, whenever P funds a new virtual channel (Fig. 17, l. 20), $[-c_{\text{virt}}]$ is appended to C and whenever P helps with the opening of a new virutal channel, but does not fund it (Fig. 17, l. 23), $[0]$ is appended to C. Therefore C consists of one list of integers for each sequence of inbound and outbound payments that have not been interrupted by a virtualisation step and a new list is added for every new virtual layer. We also observe that a non-negligent party with history (F, C) satisfies the Lemma conditions and that the value of the right hand side of the inequality (1) is equal to $\sum\limits_{s \in C} \sum\limits_{x \in s} x$, as all inbound and outbound payment values and new channel funding values that appear in the Lemma conditions are recorded in C.

Let party P with a particular history. We will inductively prove that P satisfies the Lemma. The base case is when a channel is opened with $\text{hops} = \mathcal{G}_{\text{Ledger}}$ and is closed right away, therefore $H = ([], [g])$, where $g = c$ if $P = Alice$ and $g = 0$ if $P = Bob$. P can transition to the OPEN *State* for the first time only if all of the following have taken place:

- It has received $(\text{OPEN}, c, \dots)$ while in the INIT *State*. In case $P = Alice$, this message must have been received as input by $\mathcal{E}$ (Fig. 15, l. 1), or in case $P = Bob$, this message must have been received via the network by $\bar{P}$ (Fig. 10, l. 3).
- It has received $pk_{\bar{P},F}$. In case $P = Bob$, $pk_{\bar{P},F}$ must have been contained in the $(\text{OPEN}, \dots)$ message by $\bar{P}$ (Fig. 10, l. 3), otherwise if $P = Alice$ $pk_{\bar{P},F}$ must have been contained in the $(\text{ACCEPT CHANNEL}, \dots)$ message by $\bar{P}$ (Fig. 10, l. 16).
- It internally holds a signature on the commitment transaction $C_{P,0}$ that is valid when verified with public key $pk_{\bar{P},F}$ (Fig. 12, ll. 12 and 23).
- It has the transaction F in the $\mathcal{G}_{\text{Ledger}}$ state (Fig. 13, l. 3 or Fig. 14, l. 5).

We observe that P satisfies the Lemma conditions with $m = n = l = 0$. Before transitioning to the OPEN *State*, P has produced only one valid signature for the "funding" output $(c, 2/\{pk_{P,F}, pk_{\bar{P},F}\})$ of F with $sk_{P,F}$, namely for $C_{\bar{P},0}$ (Fig. 12, ll. 4 or 14), and sent it to $\bar{P}$ (Fig. 12, ll. 6 or 21), therefore the only two ways to spend $(c, 2/\{pk_{P,F}, pk_{\bar{P},F}\})$ are by either publishing $C_{P,0}$ or $C_{\bar{P},0}$. We observe that $C_{P,0}$ has a $(g, (pk_{P,\text{out}} + (t+s)) \vee 2/\{pk_{P,R}, pk_{\bar{P},R}\})$ output (Fig. 12, l. 2 or 3). The spending method $2/\{pk_{P,R}, pk_{\bar{P},R}\}$ cannot be used since P has not produced a signature for it with $sk_{P,R}$, therefore the alternative spending method, $pk_{P,\text{out}} + (t + s)$, is the only one that will be spendable if $C_{P,0}$ is included in $\mathcal{G}_{\text{Ledger}}$, thus contributing g to the sum of outputs that contribute to inequality (1). Likewise, if $C_{\bar{P},0}$ is included in $\mathcal{G}_{\text{Ledger}}$, it will contribute at least one $(g, pk_{P,\text{out}})$ output to this inequality, as $C_{\bar{P},0}$ has a $(g, pk_{P,\text{out}})$ output (Fig. 12, l. 2 or 3). Additionally, if P receives (CLOSE) by $\mathcal{E}$ while $H = ([], [g])$, it attempts to publish $C_{P,0}$ (Fig. 26, l. 19), and will either succeed or $C_{\bar{P},0}$ will be published instead. We therefore conclude that in every case $\mathcal{G}_{\text{Ledger}}$ will eventually have a state Σ that contains at least one $(g, pk_{P,\text{out}})$ output, therefore satisfying the Lemma consequence.

Let P with history $H = (F, C)$. The induction hypothesis is that the Lemma holds for P. Let c_P the sum in the right hand side of inequality (1). In order to perform the induction step, assume that P is in the OPEN state. We will prove all the following (the facts to be proven are shown with emphasis for clarity):

- If P receives $(\text{FUND ME}, f, \dots)$ by a (local, trusted) LN ITI R, subsequently transitions back to the OPEN state (therefore moving to history (F, C') where $C' = C + [-f]$) and finally receives (CLOSE) by $\mathcal{E}$ and (CLOSED) by $\texttt{host}_P$ before any further change to its history, then *eventually P's* $\mathcal{G}_{\text{Ledger}}$ *state will contain h transaction outputs each of value* c_i *exclusively spendable or already spent by* $pk_{P,\text{out}}$*) that are descendants of* *an output with spending method* $2/\{pk_{P,F}, pk_{\bar{P},F}\}$ *such that* $\sum_{i=1}^{h} c_i \geq \sum_{s \in C'} \sum_{x \in s} x$. Furthermore, given that P moves to the OPEN state after the (FUND ME, ...) message, it also sends (FUNDED, ...) to R (Fig. 17, l. 21). If subsequently the state of R transitions to OPEN (therefore obtaining history (F_R, C_R) where $F_R = F + C$ and $C_R = [[f]]$), and finally receives (CLOSE) by $\mathcal{E}$ and (CLOSED) by $\texttt{host}_R$ ($\texttt{host}_R = \texttt{host}_P$ – Fig. 14, l. 10) before any further change to its history, then *eventually R's $\mathcal{G}_{\text{Ledger}}$ state will contain k transaction outputs each of value c_i^R exclusively spendable or already spent by $pk_{R,\text{out}}$) that are descendants of an output with spending method* $2/\{pk_{R,F}, pk_{\bar{R},F}\}$ *such that* $\sum_{i=1}^{k} c_i^R \geq \sum_{s \in C_R} \sum_{x \in s} x$.

- If P receives (VIRTUALISING, ...) by $\bar{P}$, subsequently transitions back to OPEN (therefore moving to history (F, C') where $C' = C + [0]$) and finally receives CLOSE by $\mathcal{E}$ and (CLOSED) by $\texttt{host}_P$ before any further change to its history, then *eventually P's $\mathcal{G}_{\text{Ledger}}$ state will contain h transaction outputs each of value c_i exclusively spendable or already spent by $pk_{P,\text{out}}$) that are descendants of an output with spending method* $2/\{pk_{P,F}, pk_{\bar{P},F}\}$ *such that* $\sum_{i=1}^{h} c_i \geq \sum_{s \in C} \sum_{x \in s} x$. Furthermore, given that P moves to the OPEN state after the (VIRTUALISING, ...) message and in case it sends (FUNDED, ...) to some party R (Fig. 17, l. 18), the latter party is the (local, trusted) $\texttt{fundee}$ of a new virtual channel. If subsequently the state of R transitions to OPEN (therefore obtaining history (F_R, C_R) where $F_R = F + C$ and $C_R = [[0]]$), and finally receives (CLOSE) by $\mathcal{E}$ and (CLOSED) by $\texttt{host}_R$ ($\texttt{host}_R = \texttt{host}_P$ – Fig. 14, l. 10) before any further change to its history, then *eventually R's $\mathcal{G}_{\text{Ledger}}$ state will contain an output with a* $2/\{pk_{R,F}, pk_{\bar{R},F}\}$ *spending method.*
- If P receives (PAY, d) by $\mathcal{E}$, subsequently transitions back to OPEN (therefore moving to history (F, C') where C' is C with $-d$ appended to the last list of C) and finally receives CLOSE by $\mathcal{E}$ and (CLOSED) by $\texttt{host}_P$ (the latter only if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ or equivalently $F \neq []$) before any further change to its history, then *eventually P's $\mathcal{G}_{\text{Ledger}}$ state will contain h transaction outputs each of value c_i exclusively spendable or already spent by $pk_{P,\text{out}}$) that are descendants of an output with a* $2/\{pk_{P,F}, pk_{\bar{P},F}\}$ *spending method such that* $\sum_{i=1}^{h} c_i \geq \sum_{s \in C'} \sum_{x \in s} x$.
- If P receives (GET PAID, e) by $\mathcal{E}$, subsequently transitions back to OPEN (therefore moving to history (F, C') where C' is C with e appended to the last list of C) and finally receives CLOSE by $\mathcal{E}$ and (CLOSED) by $\texttt{host}_P$ (the latter only if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ or equivalently $F = []$) before any further change to its history, then *eventually P's $\mathcal{G}_{\text{Ledger}}$ state will contain h transaction outputs each of value c_i exclusively spendable or already spent by $pk_{P,\text{out}}$) that are descendants of an output with a* $2/\{pk_{P,F}, pk_{\bar{P},F}\}$ *spending method such that* $\sum_{i=1}^{h} c_i \geq \sum_{s \in C'} \sum_{x \in s} x$.

By the induction hypothesis, before the funding procedure started P could close the channel and end up with on-chain transaction outputs exclusively spendable or already spent by $pk_{P,\text{out}}$ with a sum value of c_P. When P is in the OPEN state and receives (FUND ME, f, ...), it can only move again to the OPEN state after doing the following state transitions: OPEN → VIRTUALISING → WAITING FOR REVOCATION → WAITING FOR INBOUND REVOCATION → WAITING FOR HOSTS READY → OPEN. During this sequence of events, a new $\texttt{host}_P$ is defined (Fig. 17, l. 6), new commitment transactions are negotiated with $\bar{P}$ (Fig. 17, l. 9), control of the old funding output is handed over to $\texttt{host}_P$ (Fig. 17, l. 11), $\texttt{host}_P$ negotiates with its counterparty a new set of transactions and signatures that spend the aforementioned funding output and make available a new funding output with the keys $pk'_{P,F}, pk'_{\bar{P},F}$ as P instructed (Fig. 35 and 37) and the previous valid commitment transactions of both P and $\bar{P}$ are invalidated (Fig. 9, l. 1 and l. 14 respectively). We note that the use of the $\texttt{ANYPREVOUT}$ flag in all signatures that correspond to transaction inputs that may spend various different transaction outputs ensures that this is possible, as it avoids tying each input to a specific, predefined output. When P receives (CLOSE) by $\mathcal{E}$, it inputs (CLOSE) to $\texttt{host}_P$ (Fig. 26, l. 4). As per the Lemma conditions, $\texttt{host}_P$ will output (CLOSED). This can happen only when $\mathcal{G}_{\text{Ledger}}$ contains a suitable output for both P's and R's channel (Fig. 41, and 4 ll. 5 respectively).

If the $\texttt{host}$ of $\texttt{host}_P$ is $\mathcal{G}_{\text{Ledger}}$, then the funding output $o_{1,2} = (c_P + c_{\bar{P}}, 2/\{pk_{P,F}, pk_{\bar{P},F}\})$ for the $P, \bar{P}$ channel is already on-chain. Regarding the case in which $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$, after the funding procedure is complete, the new $\texttt{host}_P$ will have as its $\texttt{host}$ the old $\texttt{host}_P$ of P. If the (CLOSE) sequence is initiated, the new $\texttt{host}_P$ will follow the same steps that will be described below once the old $\texttt{host}_P$ succeeds in closing the lower layer (Fig. 40, l. 5). The old $\texttt{host}_P$ however will see no difference in its interface compared to what would happen if P had received (CLOSE) before the funding procedure, therefore it will successfully close by the induction hypothesis. Thereafter the process is identical to the one when the old $\texttt{host}_P = \mathcal{G}_{\text{Ledger}}$.

Moving on, $\texttt{host}_P$ is either able to publish its $\text{TX}_{1,1}$ (it has necessarily received a valid signature $\text{sig}(\text{TX}_{1,1}, pk_{\bar{P},F})$ (Fig. 37, l. 39) by its counterparty before it moved to the OPEN state for the first time), or the output $(c_P + c_{\bar{P}}, 2/\{pk_{P,F}, pk_{\bar{P},F}\})$ needed to spend $\text{TX}_{1,1}$ has already been spent. The only other transactions that can spend it are $\text{TX}_{2,1}$ and any of $(\text{TX}_{2,2,k})_{k>2}$, since these are the only transactions that spend the aforementioned output and that $\texttt{host}_P$ has signed with $sk_{P,F}$ (Fig. 37, ll. 33-37). The output can be also spent by old, revoked commitment transactions, but in that case $\texttt{host}_P$ would not have output (CLOSED); P would have instead detected this triggered by a (CHECK CHAIN FOR CLOSED) message by $\mathcal{E}$ (Fig. 24) and would have moved to the CLOSED state on its own accord (lack of such a message by $\mathcal{E}$ would lead P to become $\texttt{negligent}$, something that cannot happen according to the Lemma conditions). Every transaction among $\text{TX}_{1,1}$, $\text{TX}_{2,1}$, $(\text{TX}_{2,2,k})_{k>2}$ has a $(c_P + c_{\bar{P}} - f, 2/\{pk'_{P,F}, pk'_{\bar{P},F}\})$ output (Fig. 31, l. 18 and Fig. 30, ll. 27 and 91) which will end up in $\mathcal{G}_{\text{Ledger}}$ – call this output o_P. We will prove that at most $\sum_{i=2}^{n-1}(t_i + p + s - 1)$ blocks after (CLOSE) is received by P, an output o_R with c_{virt} coins and a $2/\{pk_{R,F}, pk_{\bar{R},F}\}$ spending condition without or with an expired timelock will be included in $\mathcal{G}_{\text{Ledger}}$. In case party $\bar{P}$ is idle, then $o_{1,2}$ is consumed by $\text{TX}_{1,1}$ and the timelock on its virtual output expires, therefore the required output o_R is on-chain. In case $\bar{P}$ is active, exactly one of $\text{TX}_{2,1}$, $(\text{TX}_{2,2,k})_{k>2}$ or $(\text{TX}_{2,3,1,k})_{k>2}$ is a descendant of $o_{1,2}$; if the transaction belongs to one of the two last transaction groups then necessarily $\text{TX}_{1,1}$ is on-chain in some block height h and given the timelock on the virtual output of $\text{TX}_{1,1}$, $\bar{P}$'s transaction can be at most at block height $h + t_2 + p + s - 1$. If $n = 3$ or $k = n - 1$, then $\bar{P}$'s unique transaction has the required output o_R (without a timelock). The rest of the cases are covered by the following sequence of events:

Closing sequence

1: `maxDel` $\leftarrow t_2 + p + s - 1$ // A_2 is active and the virtual output of $\text{TX}_{1,1}$ has a timelock of t_2
2: $i \leftarrow 3$
3: **loop**
4: **if** A_i is idle **then**
5: The timelock on the virtual output of the transaction published by A_{i-1} expires and therefore the required o_R is on-chain
6: **else** // A_i publishes a transaction that is a descendant of $o_{1,2}$
7: `maxDel` $\leftarrow$ `maxDel` $+ t_i + p + s - 1$
8: The published transaction can be of the form $\text{TX}_{i,2,2}$ or $(\text{TX}_{i,3,2,k})_{k>i}$ as it spends the virtual output which is encumbered with a public key controlled by R and R has only signed these transactions
9: **if** $i = n - 1$ or $k \geq n - 1$ **then** // The interval contains all intermediaries
10: The virtual output of the transaction is not timelocked and has only a $2/\{pk_{R,F}, pk_{\bar{R},F}\}$ spending method, therefore it is the required o_R
11: **else** // At least one intermediary is not in the interval
12: **if** the transaction is $\text{TX}_{i,3,2,k}$ **then** $i \leftarrow k$ **else** $i \leftarrow i + 1$
13: **end if**
14: **end if**
15: **end loop**
16: // `maxDel` $\leq \sum_{i=2}^{n-1}(t_i + p + s - 1)$

Figure 43

In every case o_P and o_R end up on-chain in at most s and $\sum_{i=2}^{n-1}(t_i + p + s - 1)$ blocks respectively from the moment (CLOSE) is received. The output o_P an be spent either by $C_{P,i}$ or $C_{\bar{P},i}$. Both these transactions have a $(c_P - f, pk_{P,\text{out}})$ output. This output of $C_{P,i}$ is timelocked, but the alternative spending method cannot be used as P never signed a transaction that uses it (as it is reserved for revocation, which has not taken place yet in this virtualisation layer). We have now proven that if P completes the funding of a new channel then it can close its channel for a $(c_P - f, pk_{P,\text{out}})$ output that is a descendant of an output with spending method $2/\{pk_{P,F}, pk_{\bar{P},F}\}$ and that lower bound of value holds for the duration of the funding procedure, i.e. we have proven the first claim of the first bullet.

We will now prove that the newly funded party R can close its channel securely. After R receives (FUNDED, $\texttt{host}_P, \dots$) by P and before moving to the OPEN state, it receives $\text{sig}_{\bar{R},C,0} = \text{sig}(C_{R,0}, pk_{\bar{R},F})$ and sends $\text{sig}_{R,C,0} = \text{sig}(C_{\bar{R},0}, pk_{R,F})$. Both these transactions spend o_R. As we showed before, if R receives (CLOSE) by $\mathcal{E}$ then o_R eventually ends up on-chain. After receiving (CLOSED) from $\texttt{host}_P$, R attempts to add $C_{R,0}$ to $\mathcal{G}_{\text{Ledger}}$, which may only fail if $C_{\bar{R},0}$ ends up on-chain instead. Similar to the case of P, both these transactions have an $(f, pk_{R,\text{out}})$ output. This output of $C_{R,0}$ is timelocked, but the alternative spending method cannot be used as R never signed a transaction that uses it (as it is reserved for revocation, which has not taken place yet) so the timelock will expire and the desired spending method will be available. We have now proven that if R's channel is funded to completion (i.e. R moves to the OPEN state for the first time) then it can close its channel for a $(f, pk_{R,\text{out}})$ output that is a descendant of o_R. We have therefore proven the first bullet.

We now move on to the second bullet. In case P is the `funder` (i.e. $i = n$), then the same arguments as in the previous bullet hold here with "WAITING FOR INBOUND REVOCATION" replaced with "WAITING FOR OUTBOUND REVOCATION", $o_{1,2}$ with $o_{n-1,n}$, $\text{TX}_{1,1}$ with $\text{TX}_{n,1}$, $\text{TX}_{2,1}$ with $\text{TX}_{n-1,1}$, $(\text{TX}_{2,2,k})_{k>2}$ with $(\text{TX}_{n-1,2,k})_{k<n-1}$, $(\text{TX}_{2,3,1,k})_{k>2}$ with $(\text{TX}_{n-1,3,n,k})_{k<n-1}$, t_2 with t_{n-1}, $\text{TX}_{i,3,2,k}$ with $\text{TX}_{i,3,n-1,k}$, i is initialized to $n - 2$ in l. 2 of Fig. 43, i is decremented instead of incremented in l. 12 of the same Figure and f is replaced with 0. This is so because these two cases are symmetric.

In case P is not the `funder` $(1 < i < n)$, then we only need to prove the first statement of the second bullet. By the induction hypothesis and since `sibling` is trusted, we know that both P's and `sibling`'s funding outputs either are or can be eventually put on-chain and that P's funding output has at least $c_P = \sum_{s \in C} \sum_{x \in s} x$ coins. If P is on the "left" of its `sibling` (i.e. there is an untrusted party that sent the (VIRTUALISING, ...) message to P which triggered the latter to move to the VIRTUALISING state and to send a (VIRTUALISING, ...) message to its own `sibling`), the "left" funding output o_{left} (the one held with the untrusted party to the left) can be spent by one of $\text{TX}_{i,1}$, $(\text{TX}_{i,2,k})_{k>i}$, $\text{TX}_{i-1,1}$, or $(\text{TX}_{i-1,2,k})_{k<i-1}$, as these are the only transactions that P has signed with $sk_{P,F}$. All these transactions have a $(c_P + c_{\bar{P}} - f, 2/\{pk_{P,F'}, pk_{\bar{P},F'}\})$ output that can in turn be spent by either $C_{P,0}$ or $C_{\bar{P},0}$, both of which have an output of value c_P and a $pk_{P,\text{out}}$ spending method and no other spending method can be used (as P has not signed the "revocation" spending method of $C_{P,0}$).

In the case that P is to the right of its `sibling` (i.e. P receives by `sibling` the (VIRTUALISING, ...) message that causes P's transition to the VIRUTALISING state), the "right" funding output o_{right} (the one held with the untrusted party to the right) can be spent by one of $\text{TX}_{i,1}$, $(\text{TX}_{i,2,k})_{k<i}$, $\text{TX}_{i+1,1}$, or $(\text{TX}_{i+1,2,k})_{k>i+1}$, as these are the only transactions that P has signed with $sk_{P,F}$. All these transactions have a $(c_P + c_{\bar{P}} - f, 2/\{pk_{P,F'}, pk_{\bar{P},F'}\})$ output that can in turn be spent by either $C_{P,0}$ or $C_{\bar{P},0}$, both of which have an output of value $c_P - f$ and a $pk_{P,\text{out}}$ spending method and no other spending method can be used (as P has not signed the "revocation" spending method of $C_{P,0}$). P can get the remaining f coins as follows: $\text{TX}_{i,1}$ and all of $(\text{TX}_{i,2,k})_{k<i}$ already have an $(f, pk_{P,\text{out}})$ output. If instead $\text{TX}_{i+1,1}$ or

one of $(\mathrm{TX}_{i+1,2,k_2})_{k_2>i+1}$ spends o_{right}, then P will publish $\mathrm{TX}_{i,2,i+1}$ or $\mathrm{TX}_{i,2,k_2}$ respectively if o_{left} is unspent, otherwise o_{left} is spent by one of $\mathrm{TX}_{i-1,1}$ or $(\mathrm{TX}_{i-1,2,k_1})_{k_1<i-1}$ in which case P will publish one of $\mathrm{TX}_{i,3,k_1,i+1}$, $\mathrm{TX}_{i,3,i-1,k_2}$, $\mathrm{TX}_{i,3,i-1,i+1}$ or $\mathrm{TX}_{i,3,k_1,k_2}$. In particular, $\mathrm{TX}_{i,3,k_1,i+1}$ is published if $\mathrm{TX}_{i-1,2,k_1}$ and $\mathrm{TX}_{i+1,1}$ are on-chain, $\mathrm{TX}_{i,3,i-1,k_2}$ is published if $\mathrm{TX}_{i-1,1}$ and $\mathrm{TX}_{i+1,2,k_2}$ are on-chain, $\mathrm{TX}_{i,3,i-1,i+1}$ is published if $\mathrm{TX}_{i-1,1}$ and $\mathrm{TX}_{i+1,1}$ are on-chain, or $\mathrm{TX}_{i,3,k_1,k_2}$ is published if $\mathrm{TX}_{i-1,2,k_1}$ and $\mathrm{TX}_{i+1,2,k_2}$ are on-chain. All these transactions include an $(f, pk_{P,\text{out}})$ output. We have therefore covered all cases and proven the second bullet.

Regarding now the third bullet, once again the induction hypothesis guarantees that before (PAY, d) was received, P could close the channel resulting in on-chain outputs exclusively spendable or already spent by $pk_{P,\text{out}}$ that are descendants of an output with a $2/\{pk_{P,F}, pk_{\bar{P},F}\}$ spending method that have a sum value of $c_P = \sum\limits_{s \in C} \sum\limits_{x \in s} x$. (Note that $\sum\limits_{s \in C'} \sum\limits_{x \in s} x = d + \sum\limits_{s \in C} \sum\limits_{x \in s} x$.) When P receives (PAY, d) while in the OPEN state, it moves to the WAITING FOR COMMITMENT SIGNED state before returning to the OPEN state. It signs (Fig. 19, l. 2) the new commitment transaction $C_{\bar{P},i+1}$ in which the counterparty owns d more coins than before that moment (Fig. 19, l. 1), sends the signature to the counterparty (Fig. 19, l. 4) and expects valid signatures on its own updated commitment transaction (Fig. 20, l. 1) and the revocation transaction for the old commitment transaction of the counterparty (Fig. 20, l. 3). Note that if the counterparty does not respond or if it responds with missing/invalid signatures, either P can close the channel with the old commitment transaction $C_{P,i}$ exactly like before the update started (as it has not yet sent the signature for the old revocation transaction), or the counterparty will close the channel either with the new or with the old commitment transaction. In all cases in which validation fails and the channel closes, there is an output with a $pk_{P,\text{out}}$ spending method and no other useable spending method that carries at least $c_P - d$ coins. Only if the verification succeeds does P sign (Fig. 20, l. 5) and send (Fig. 20, l. 17) the counterparty's revocation transaction for P's previous commitment transaction.

Similarly to previous bullets, if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ the funding output can be put on-chain, otherwise the funding output is already on-chain. In both cases, since the closing procedure continues, one of $C_{P,i+1}$ $(C_{\bar{P},j})_{0 \leq j \leq i+1}$ will end up on-chain. If $C_{\bar{P},j}$ for some $j < i+1$ is on-chain, then P submits $R_{P,j}$ (we discussed how P obtained $R_{P,i}$ and the rest of the cases are covered by induction) and takes the entire value of the channel which is at least $c_P - d$. If $C_{\bar{P},i+1}$ is on-chain, it has a $(c_P - d, pk_{P,\text{out}})$ output. If $C_{P,i+1}$ is on-chain, it has an output of value $c_P - d$, a timelocked $pk_{P,\text{out}}$ spending method and a non-timelocked spending method that needs the signature made with $sk_{P,R}$ on $R_{\bar{P},i+1}$. P however has not generated that signature, therefore this spending method cannot be used and the timelock will expire, therefore in all cases outputs that descend from the funding output, can be spent exclusively by $pk_{P,\text{out}}$ and carry at least $c_P - d$ coins are put on-chain. We have proven the third bullet.

For the fourth and last bullet, again by the induction hypothesis, before (GET PAID, e) was received P could close the channel resulting in on-chain outputs exclusively spendable or already spent by $pk_{P,\text{out}}$ that are descendants of an output o_F with a $2/\{pk_{P,F}, pk_{\bar{P},F}\}$ spending method and have a sum value of $c_P = \sum\limits_{s \in C} \sum\limits_{x \in s} x$. (Note that $e + \sum\limits_{s \in C'} \sum\limits_{x \in s} x = \sum\limits_{s \in C} \sum\limits_{x \in s} x$ and that o_F either is already on-chain or can be eventually put on-chain as we have argued in the previous bullets by the induction hypothesis.) When P receives (GET PAID, e) while in the OPEN state, if the balance of the counterparty is enough it moves to the WAITING TO GET PAID state (Fig. 22, l. 6). If subsequently it receives a valid signature for $C_{P,i+1}$ (Fig. 19, l. 8) which is a commitment transaction that can spend the o_F output and gives to P an additional e coins compared to $C_{P,i}$. Subsequently P's state transitions to WAITING FOR PAY REVOCATION and sends signatures for $C_{\bar{P},i+1}$ and $R_{\bar{P},i}$ to $\bar{P}$. If the o_F output is spent while P is in the latter state, it can be spent by one of $C_{P,i+1}$ or $(C_{\bar{P},j})_{0 \leq j \leq i+1}$. If it is spent by $C_{P,i+1}$ or $C_{\bar{P},i+1}$, then these two transactions have a $(c_P + e, pk_{P,\text{out}})$ output. (Note that the former is encumbered with a timelock, but the alternative spending method cannot be used as P has not signed $R_{\bar{P},i+1}$.) If it is spent by $C_{\bar{P},i}$ then a $(c_P, pk_{P,\text{out}})$ output becomes available instead, therefore P can still get the c_P coins that correspond to the previous state. If any of $(C_{\bar{P},j})_{0 \leq j < i}$ spends o_F then it makes available a $pk_{P,\text{out}}$ output with the coins that P had at state j and additionally P can publish $R_{P,j}$ that spends $\bar{P}$'s output of $C_{\bar{P},j}$ and obtain the entirety of $\bar{P}$'s coins at state j for a total of $c_P + c_{\bar{P}}$ coins. Therefore in every case P can claim at least c_P coins. In the case that P instead subsequently receives a valid signature to $R_{P,i}$ (Fig. 20, l. 20) it finally moves to the OPEN state once again. In this state the above analysis of what can happen when o_F holds similarly, with the difference that if $\bar{P}$ spends o_F with $C_{\bar{P},i}$ now P can publish $R_{P,i}$ which gives P the coins of $\bar{P}$. Therefore with this difference P is now guaranteed to gain at least $c_P + e$ coins upon channel closure. We have therefore proven the fourth bullet.

LEMMA 8.2 (IDEAL WORLD BALANCE). *Consider an ideal world execution with functionality $\mathcal{F}_{\text{Chan}}$ and simulator $\mathcal{S}$. Let $P \in \{Alice, Bob\}$ one of the two parties of $\mathcal{F}_{\text{Chan}}$. Assume that all of the following are true:*

- *$State_P \neq$ IGNORED,*
- *P has transitioned to the OPEN State at least once. Additionally, if $P = Alice$, it has received (OPEN, $c, \ldots$) by $\mathcal{E}$ prior to transitioning to the OPEN State,*
- *P [has received (FUND ME, $f_i, \ldots$) as input by another $\mathcal{F}_{\text{Chan}}$/LN ITI while $State_P =$ OPEN and P subsequently transitioned to OPEN State] $n \geq 0$ times,*
- *P [has received (PAY, d_i) by $\mathcal{E}$ while $State_P =$ OPEN and P subsequently transitioned to OPEN State] $m \geq 0$ times,*
- *P [has received (GET PAID, e_i) by $\mathcal{E}$ while $State_P =$ OPEN and P subsequently transitioned to OPEN State] $l \geq 0$ times.*

Let $\phi = 1$ if $P = Alice$, or $\phi = 0$ if $P = Bob$. If $\mathcal{F}_{\text{Chan}}$ receives (CLOSE, P) by $\mathcal{S}$, then the following holds with overwhelming probability on the security parameter:

$$\texttt{balance}_P = \phi \cdot c - \sum_{i=1}^{n} f_i - \sum_{i=1}^{m} d_i + \sum_{i=1}^{l} e_i \tag{2}$$

We will prove the Lemma by following the evolution of the $\texttt{balance}_P$ variable.

- When $\mathcal{F}_{\text{Chan}}$ is activated for the first time, it sets $\texttt{balance}_P \leftarrow 0$ (Fig. 2, l. 1).
- If $P = Alice$ and it receives (OPEN, $c, \ldots$) by $\mathcal{E}$, it stores c (Fig. 2, l. 10). If later $State_P$ becomes OPEN, $\mathcal{F}_{\text{Chan}}$ sets

$\texttt{balance}_P \leftarrow c$ (Fig. 2, ll. 13 or 31). In contrast, if $P = Bob$, it is $\texttt{balance}_P = 0$ until at least the first transition of $State_P$ to OPEN (Fig. 2).

- Every time P receives input (FUND ME, $f_i, \dots$) by another party while $State_P =$ OPEN, P stores f_i (Fig. 4, l. 1). The next time $State_P$ transitions to OPEN (if such a transition happens), $\texttt{balance}_P$ is decremented by f_i (Fig. 4, l. 27). Therefore, if this cycle happens $n \geq 0$ times, $\texttt{balance}_P$ will be decremented by $\sum_{i=1}^{n} f_i$ in total.
- Every time P receives input (PAY, d_i) by $\mathcal{E}$ while $State_P =$ OPEN, d_i is stored (Fig. 3, l. 2). The next time $State_P$ transitions to OPEN (if such a transition happens), $\texttt{balance}_P$ is decremented by d_i (Fig. 3, l. 13). Therefore, if this cycle happens $m \geq 0$ times, $\texttt{balance}_P$ will be decremented by $\sum_{i=1}^{m} d_i$ in total.
- Every time P receives input (GET PAID, e_i) by $\mathcal{E}$ while $State_P =$ OPEN, e_i is stored (Fig. 3, l. 7). The next time $State_P$ transitions to OPEN (if such a transition happens) $\texttt{balance}_P$ is incremented by e_i (Fig. 3, l. 19). Therefore, if this cycle happens $l \geq 0$ times, $\texttt{balance}_P$ will be incremented by $\sum_{i=1}^{l} e_i$ in total.

On aggregate, after the above are completed and then $\mathcal{F}_{\text{Chan}}$ receives (CLOSE, P) by $\mathcal{S}$, it is $\texttt{balance}_P = c - \sum_{i=1}^{n} f_i - \sum_{i=1}^{m} d_i + \sum_{i=1}^{l} e_i$ if $P = Alice$, or else if $P = Bob$, $\texttt{balance}_P = -\sum_{i=1}^{n} f_i - \sum_{i=1}^{m} d_i + \sum_{i=1}^{l} e_i$.

LEMMA 8.3 (NO HALT). *In an ideal execution with $\mathcal{F}_{\text{Chan}}$ and $\mathcal{S}$, if the trusted parties of the honest parties of $\mathcal{F}_{\text{Chan}}$ are themselves honest, then the functionality halts with negligible probability in the security parameter (i.e. l. 21 of Fig. 5 is executed negligibly often).*

We prove the Lemma in two steps. We first show that if the conditions of Lemma 8.2 hold, then the conditions of Lemma 8.1 for the real world execution with protocol LN and the same $\mathcal{E}$ and $\mathcal{A}$ hold as well for the same m, n and l values.

For $State_P$ to become IGNORED, either $\mathcal{S}$ has to send (BECAME CORRUPTED OR NEGLIGENT, P) or $\texttt{host}_P$ must output (ENABLER USED REVOCATION) to $\mathcal{F}_{\text{Chan}}$ (Fig. 2, l. 4). The first case only happens when either P receives (CORRUPT) by $\mathcal{A}$ (Fig. 7, l. 1), which means that the simulated P is not honest anymore, or when P becomes $\texttt{negligent}$ (Fig. 7, l. 4), which means that the first condition of Lemma 8.1 is violated. In the second case, it is $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ and the state of $\texttt{host}_P$ is GUEST PUNISHED (Fig. 42, ll. 1 or 12), so in case P receives (CLOSE) by $\mathcal{E}$ the output of $\texttt{host}_P$ will be (GUEST PUNISHED) (Fig. 40, l. 3). In all cases, some condition of Lemma 8.1 is violated.

For $State_P$ to become OPEN at least once, the following sequence of events must take place (Fig. 2): If $P = Alice$, it must receive (INIT, pk) by $\mathcal{E}$ when $State_P =$ UNINIT, then either receive (OPEN, c, $\mathcal{G}_{\text{Ledger}}, \dots$) by $\mathcal{E}$ and (BASE OPEN) by $\mathcal{S}$ or (OPEN, c, $\texttt{hops}$ ($\neq \mathcal{G}_{\text{Ledger}}$), $\dots$) by $\mathcal{E}$, (FUNDED, HOST, $\dots$) by $\texttt{hops[0].left}$ and (VIRTUAL OPEN) by $\mathcal{S}$. In either case, $\mathcal{S}$ only sends its message only if all its simulated honest parties move to the OPEN state (Fig. 7, l. 10), therefore if the second condition of Lemma 8.2 holds and $P = Alice$, then the second condition of Lemma 8.1 holds as well. The same line of reasoning can be used to deduce that if $P = Bob$, then $State_P$ will become OPEN for the first time only if all honest simulated parties move to the OPEN *state*, therefore once more the second condition of Lemma 8.2 holds only if the second condition of Lemma 8.1 holds as well. We also observe that, if both parties are honest, they will transition to the OPEN state simultaneously.

Regarding the third Lemma 8.2 condition, we assume (and will later show) that if both parties are honest and the state of one is OPEN, then the state of the other is also OPEN. Each time P receives input (FUND ME, $f, \dots$) by $R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$, $State_P$ transitions to PENDING FUND, subsequently when a command to define a new VIRT ITI through P is intercepted by $\mathcal{F}_{\text{Chan}}$, $State_P$ transitions to TENTATIVE FUND and afterwards when $\mathcal{S}$ sends (FUND) to $\mathcal{F}_{\text{Chan}}$, $State_P$ transitions to SYNC FUND. In parallel, if $State_{\bar{P}} =$ IGNORED, then $State_P$ transitions directly back to OPEN. If on the other hand $State_{\bar{P}} =$ OPEN and $\mathcal{F}_{\text{Chan}}$ intercepts a similar VIRT ITI definition command through $\bar{P}$, $State_{\bar{P}}$ transitions to TENTATIVE HELP FUND. On receiving the aforementioned (FUND) message by $\mathcal{S}$ and given that $State_{\bar{P}} =$ TENTATIVE HELP FUND, $\mathcal{F}_{\text{Chan}}$ also sets $State_{\bar{P}}$ to SYNC HELP FUND. Then both $State_{\bar{P}}$ and $State_P$ transition simultaneously to OPEN (Fig. 4). This sequence of events may repeat any $n \geq 0$ times. We observe that throughout these steps, honest simulated P has received (FUND ME, $f, \dots$) and that $\mathcal{S}$ only sends (FUND) when all honest simulated parties have transitioned to the OPEN state (Fig. 7, l. 18 and Fig. 17, l. 12), so the third condition of Lemma 8.1 holds with the same n as that of Lemma 8.2.

Regarding the fourth Lemma 8.2 condition, we again assume that if both parties are honest and the state of one is OPEN, then the state of the other is also OPEN. Each time P receives input (PAY, d) by $\mathcal{E}$, $State_P$ tranisitions to TENTATIVE PAY and subsequently when $\mathcal{S}$ sends (PAY) to $\mathcal{F}_{\text{Chan}}$, $State_P$ transitions to (SYNC PAY, d). In parallel, if $State_{\bar{P}} =$ IGNORED, then $State_P$ transitions directly back to OPEN. If on the other hand $State_{\bar{P}} =$ OPEN and $\mathcal{F}_{\text{Chan}}$ receives (GET PAID, d) by $\mathcal{E}$ addressed to $\bar{P}$, $State_{\bar{P}}$ transitions to TENTATIVE GET PAID. On receiving the aforementioned (PAY) message by $\mathcal{S}$ and given that $State_{\bar{P}} =$ TENTATIVE GET PAID, $\mathcal{F}_{\text{Chan}}$ also sets $State_{\bar{P}}$ to SYNC GET PAID. Then both $State_P$ and $State_{\bar{P}}$ transition simultaneously to OPEN (Fig. 3). This sequence of events may repeat any $m \geq 0$ times. We observe that throughout these steps, honest simulated P has received (PAY, d) and that $\mathcal{S}$ only sends (PAY) when all honest simulated parties have completed sending or receiving the payment (Fig. 7, l. 16), so the fourth condition of Lemma 8.1 holds with the same m as that of Lemma 8.2. As far as the fifth condition of Lemma 8.2 goes, we observe that this case is symmetric to the one discussed for its fourth condition above if we swap P and $\bar{P}$, therefore we deduce that if Lemma 8.2 holds with some l, then Lemma 8.1 holds with the same l.

As promised, we here argue that if both parties are honest and one party moves to the OPEN state, then the other party will move to the OPEN state as well. We already saw that the first time one party moves to the OPEN state, it will happen simultaneously with the same transition for the other party. We also saw that, when a party transitions from the SYNC HELP FUND or the SYNC FUND state to the OPEN state, then the other party will also transition to the OPEN state simultaneously. Furthermore, we saw that if one party transitions from the SYNC PAY or the SYNC GET PAID state to the OPEN state, the other party will also transition to the OPEN state simultaneously. Lastly we notice that we have exhausted all manners in which a party can transition to the OPEN state, therefore we have proven that transitions of honest parties to the OPEN state happen simultaneously.

Now, given that $\mathcal{S}$ internally simulates faithfully both LN parties and that $\mathcal{F}_{\text{Chan}}$ relinquishes to $\mathcal{S}$ complete control of the external communication of the parties as long as it does not halt, we deduce that $\mathcal{S}$ replicates the behaviour of the aforementioned real world. By combining these facts with the consequences of the two Lemmas and the check that leads $\mathcal{F}_{\text{Chan}}$ to halt if it fails (Fig. 5, l. 18), we deduce that if the conditions of Lemma 8.2 hold for the honest parties of $\mathcal{F}_{\text{Chan}}$ and their trusted parties, then the functionality halts only with negligible probability.

In the second proof step, we show that if the conditions of Lemma 8.2 do not hold, then the check of Fig. 5, l. 18 never takes place. We first discuss the $State_P$ = IGNORED case. We observe that the IGNORED *State* is a sink state, as there is no way to leave it once in. Additionally, for the balance check to happen, $\mathcal{F}_{\text{Chan}}$ must receive (CLOSED, P) by $\mathcal{S}$ when $State_P \neq$ IGNORED (Fig. 5, l. 9). We deduce that, once $State_P$ = IGNORED, the balance check will not happen. Moving to the case where $State_P$ has never been OPEN, we observe that it is impossible to move to any of the states required by l. 9 of Fig. 5 without first having been in the OPEN state. Moreover if $P = Alice$, it is impossible to reach the OPEN state without receiving input (OPEN, $c, \dots$) by $\mathcal{E}$. Lastly, as we have observed already, the three last conditions of Lemma 8.2 are always satisfied. We conclude that if the conditions to Lemma 8.2 do not hold, then the check of Fig. 5, l. 18 does not happen and therefore $\mathcal{F}_{\text{Chan}}$ does not halt.

On aggregate, $\mathcal{F}_{\text{Chan}}$ may only halt with negligible probability in the security parameter.

THEOREM 8.4 (RECURSIVE VIRTUAL PAYMENT CHANNEL SECURITY). *The protocol Π_{Chan} UC-realises $\mathcal{F}_{\text{Chan}}$ given a global functionality $\mathcal{G}_{\text{Ledger}}$ and assuming the security of the underlying digital signature. Specifically,*

$$\forall \textit{ PPT } \mathcal{A}, \exists \textit{ PPT } \mathcal{S} : \forall \textit{ PPT } \mathcal{E} \textit{ it is } \textsc{exec}^{\mathcal{G}_{\text{Ledger}}}_{\Pi_{\text{Chan}},\mathcal{A},\mathcal{E}} \approx \textsc{exec}^{\mathcal{F}_{\text{Chan}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S},\mathcal{E}}$$

By inspection of Figs. 1 and 6 we can deduce that for a particular $\mathcal{E}$, in the ideal world execution $\textsc{exec}^{\mathcal{F}_{\text{Chan}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S}_{\mathcal{A}},\mathcal{E}}$, $\mathcal{S}_{\mathcal{A}}$ simulates internally the two Π_{Chan} parties exactly as they would execute in the real world execution, $\textsc{exec}^{\mathcal{G}_{\text{Ledger}}}_{\Pi_{\text{Chan}},\mathcal{A},\mathcal{E}}$ in case $\mathcal{F}_{\text{Chan}}$ does not halt. Indeed, $\mathcal{F}_{\text{Chan}}$ only halts with negligible probability according to Lemma 8.3, therefore the two executions are computationally indistinguishable.

We now generalise Theorem 8.4 to prove the indistinguishability of multiple $\mathcal{F}_{\text{Chan}}$ instances from multiple Π_{Chan} instances, leveraging the definition of the multi-session extension of an ideal functionality [37].

Definition 8.5 (Multi-Session Extension of a Protocol). Let protocol π. Its *multi-session extension* $\widehat{\pi}$ has the same code as π and has 2 session ids: the "sub-session id" *ssid* which replaces the session id of π and the usual session id *sid* which has no further function apart from what is prescribed by the UC framework.

THEOREM 8.6 (INDISTINGUISHABILITY OF MULTIPLE SESSIONS). *Let $\widehat{\mathcal{F}}_{\text{Chan}}$ the multi-session extension of $\mathcal{F}_{\text{Chan}}$ and $\widehat{\Pi}_{\text{Chan}}$ the protocol-multi-session extension of Π_{Chan}.*

$$\forall \textit{ PPT } \mathcal{A}, \exists \textit{ PPT } \mathcal{S} : \forall \textit{ PPT } \mathcal{E} \textit{ it is } \textsc{exec}^{\mathcal{G}_{\text{Ledger}}}_{\widehat{\Pi}_{\text{Chan}},\mathcal{A},\mathcal{E}} \approx \textsc{exec}^{\widehat{\mathcal{F}}_{\text{Chan}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S},\mathcal{E}}$$

We observe that $\widehat{\mathcal{F}}_{\text{Chan}}$ uses $\mathcal{F}_{\text{Chan}}$ internally. According to the UC theorem [4] and given that Π_{Chan} UC-realises $\mathcal{F}_{\text{Chan}}$ (Theorem 8.4), $\widehat{\mathcal{F}}^{\mathcal{F}_{\text{Chan}} \to \Pi_{\text{Chan}}}_{\text{Chan}}$ UC-emulates $\widehat{\mathcal{F}}_{\text{Chan}}$. We now observe that $\widehat{\mathcal{F}}^{\mathcal{F}_{\text{Chan}} \to \Pi_{\text{Chan}}}_{\text{Chan}}$ behaves identically to a session with $\widehat{\Pi}_{\text{Chan}}$ protocols, as the former routes each message to the same internal Π_{Chan} instance that would handle the same message in the latter case, therefore $\widehat{\mathcal{F}}^{\mathcal{F}_{\text{Chan}} \to \Pi_{\text{Chan}}}_{\text{Chan}}$ UC-emulates $\widehat{\Pi}_{\text{Chan}}$. By the transitivity of UC-emulation, we deduce that $\widehat{\mathcal{F}}_{\text{Chan}}$ UC-emulates $\widehat{\Pi}_{\text{Chan}}$.

REFERENCES

[1] Nakamoto S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008)
[2] Croman K., Decker C., Eyal I., Gencer A. E., Juels A., Kosba A., Miller A., Saxena P., Shi E., Sirer E. G., et al.: On scaling decentralized blockchains. In International Conference on Financial Cryptography and Data Security: pp. 106–125: Springer (2016)
[3] Poon J., Dryja T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf (2016)
[4] Canetti R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA: pp. 136–145: doi:10.1109/SFCS.2001.959888: URL https://eprint.iacr.org/2000/067.pdf (2001)
[5] Gudgeon L., Moreno-Sanchez P., Roos S., McCorry P., Gervais A.: SoK: Layer-Two Blockchain Protocols. In Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers: pp. 201–226: doi:10.1007/978-3-030-51280-4_12: URL https://doi.org/10.1007/978-3-030-51280-4_12 (2020)
[6] Decker C., Wattenhofer R.: A fast and scalable payment network with bitcoin duplex micropayment channels. In Symposium on Self-Stabilizing Systems: pp. 3–18: Springer (2015)
[7] Wood G.: Ethereum: A secure decentralised generalised transaction ledger
[8] Raiden Network. https://raiden.network/
[9] Malavolta G., Moreno-Sanchez P., Schneidewind C., Kate A., Maffei M.: Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019 (2019)
[10] Aumayr L., Ersoy O., Erwig A., Faust S., Hostakova K., Maffei M., Moreno-Sanchez P., Riahi S.: Generalized Bitcoin-Compatible Channels. Cryptology ePrint Archive, Report 2020/476: https://eprint.iacr.org/2020/476 (2020)
[11] Sivaraman V., Venkatakrishnan S. B., Alizadeh M., Fanti G. C., Viswanath P.: Routing Cryptocurrency with the Spider Network. CoRR: vol. abs/1809.05088: URL http://arxiv.org/abs/1809.05088 (2018)
[12] Prihodko P., Zhigulin S., Sahno M., Ostrovskiy A., Osuntokun O.: Flare: An approach to routing in lightning network. White Paper (2016)
[13] Miller A., Bentov I., Kumaresan R., Cordi C., McCorry P.: Sprites and State Channels: Payment Networks that Go Faster than Lightning. ArXiv preprint arXiv:1702.05812 (2017)
[14] Dziembowski S., Eckey L., Faust S., Malinowski D.: Perun: Virtual Payment Hubs over Cryptocurrencies. In 2019 2019 IEEE Symposium on Security and Privacy (SP): pp. 344–361: IEEE Computer Society, Los Alamitos, CA, USA: ISSN 2375–1207: doi:10.1109/SP.2019.00020: URL https://doi.ieeecomputersociety.org/10.1109/SP.2019.00020 (2019)
[15] Dziembowski S., Faust S., Hostáková K.: General State Channel Networks. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018: pp. 949–966: doi:10.1145/3243734.3243856: URL https://doi.org/10.1145/3243734.3243856 (2018)
[16] Dong M., Liang Q., Li X., Liu J.: Celer Network: Bring Internet Scale to Every Blockchain (2018)
[17] Chakravarty M. M. T., Coretti S., Fitzi M., Gazi P., Kant P., Kiayias A., Russell A.: Hydra: Fast Isomorphic State Channels. Cryptology ePrint Archive, Report 2020/299: https://eprint.iacr.org/2020/299 (2020)
[18] Chakravarty M. M. T., Kireev R., MacKenzie K., McHale V., Müller J., Nemish A., Nester C., Peyton Jones M., Thompson S., Valentine R., Wadler P.: Functional blockchain contracts. https://iohk.io/en/research/library/papers/functional-blockchain-contracts/ (2019)

[19] Burchert C., Decker C., Wattenhofer R.: Scalable funding of Bitcoin micropayment channel networks. In The Royal Society: doi:10.1098/rsos.180089 (2018)

[20] Egger C., Moreno-Sanchez P., Maffei M.: Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security: CCS '19: p. 801–815: Association for Computing Machinery, New York, NY, USA: ISBN 9781450367479: doi:10.1145/3319535.3345666: URL https://doi.org/10.1145/3319535.3345666 (2019)

[21] Zhao L., Shuang H., Xu S., Huang W., Cui R., Bettadpur P., Lie D.: SoK: Hardware Security Support for Trustworthy Execution (2019)

[22] Lind J., Eyal I., Pietzuch P. R., Sirer E. G.: Teechan: Payment Channels Using Trusted Execution Environments. CoRR: vol. abs/1612.07766: URL http://arxiv.org/abs/1612.07766 (2016)

[23] Lind J., Naor O., Eyal I., Kelbert F., Sirer E. G., Pietzuch P.: Teechain: A Secure Payment Network with Asynchronous Blockchain Access. In Proceedings of the 27th ACM Symposium on Operating Systems Principles: SOSP '19: p. 63–79: Association for Computing Machinery, New York, NY, USA: ISBN 9781450368735: doi:10.1145/3341301.3359627: URL https://doi.org/10.1145/3341301.3359627 (2019)

[24] Liao J., Zhang F., Sun W., Shi W.: Speedster: A TEE-assisted State Channel System (2021)

[25] Avarikioti G., Kogias E. K., Wattenhofer R., Zindros D.: Brick: Asynchronous Payment Channels (2020)

[26] Dziembowski S., Fabiański G., Faust S., Riahi S.: Lower Bounds for Off-Chain Protocols: Exploring the Limits of Plasma. Cryptology ePrint Archive, Report 2020/175: https://eprint.iacr.org/2020/175 (2020)

[27] Armknecht F., Karame G. O., Mandal A., Youssef F., Zenner E.: Ripple: Overview and Outlook. In Trust and Trustworthy Computing - 8th International Conference, TRUST 2015, Heraklion, Greece, August 24-26, 2015, Proceedings: pp. 163–180: doi:10.1007/978-3-319-22846-4_10: URL https://doi.org/10.1007/978-3-319-22846-4_10 (2015)

[28] Mazieres D.: The stellar consensus protocol: A federated model for internet-level consensus. Stellar Development Foundation (2015)

[29] Malavolta G., Moreno-Sanchez P., Kate A., Maffei M.: SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks (2016)

[30] Roos S., Moreno-Sanchez P., Kate A., Goldberg I.: Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions. In 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018: URL http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_09-3_Roos_paper.pdf (2018)

[31] Jourenko M., Larangeira M., Tanaka K.: Lightweight Virtual Payment Channels. In S. Krenn, H. Shulman, S. Vaudenay (editors), Cryptology and Network Security: pp. 365–384: Springer International Publishing, Cham: ISBN 978-3-030-65411-5 (2020)

[32] Aumayr L., Ersoy O., Erwig A., Faust S., Hostáková K., Maffei M., Moreno-Sanchez P., Riahi S.: Bitcoin-Compatible Virtual Channels. In IEEE Symposium on Security and Privacy, Oakland, USA; 2021-05-23 - 2021-05-27: https://eprint.iacr.org/2020/554.pdf (2021)

[33] Decker C., Towns A.: SIGHASH_ANYPREVOUT for Taproot Scripts. https://github.com/bitcoin/bips/blob/master/bip-0118.mediawiki

[34] Lindell Y.: How to Simulate It - A Tutorial on the Simulation Proof Technique. In Tutorials on the Foundations of Cryptography: pp. 277–346: doi:10.1007/978-3-319-57048-8_6: URL https://doi.org/10.1007/978-3-319-57048-8_6 (2017)

[35] Kiayias A., Litos O. S. T.: A Composable Security Treatment of the Lightning Network. In 33rd IEEE Computer Security Foundations Symposium, CSF 2020, Boston, MA, USA, June 22-26, 2020: pp. 334–349: doi:10.1109/CSF49147.2020.00031: URL https://doi.org/10.1109/CSF49147.2020.00031 (2020)

[36] Decker C., Russell R., Osuntokun O.: eltoo: A Simple Layer2 Protocol for Bitcoin. https://blockstream.com/eltoo.pdf

[37] Canetti R., Rabin T.: Universal Composition with Joint State. In Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings: pp. 265–281: doi:10.1007/978-3-540-45146-4_16: URL https://doi.org/10.1007/978-3-540-45146-4_16 (2003)

[38] Badertscher C., Maurer U., Tschudi D., Zikas V.: Bitcoin as a transaction ledger: A composable treatment. In Annual International Cryptology Conference: pp. 324–356: Springer (2017)