TODO: Add support for cooperative adding multiple virtuals to single channel as future work (needs cooperation by all hops of all existing virtuals of current channel) TODO: Add support for cooperative closing as future work

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

- On receiving (`msg`) by party R to $P \in \{Alice, Bob\}$ by means of `mode` $\in \{\text{input}, \text{output}, \text{network}\}$, 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` $\in \{\text{input}, \text{output}, \text{network}\}$, $P \in \{Alice, Bob\}$), 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

Fig. 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
    $\texttt{enabler}_A \leftarrow \texttt{hops}[0]\texttt{.left}$
    $State_A \leftarrow$ PENDING VIRTUAL OPEN

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

On output (FUNDED, $\texttt{host}, \dots$) to Bob by ITI $R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$ when $State_B =$ INIT:
    $\texttt{enabler}_B \leftarrow R$
    $\texttt{host}_B \leftarrow \texttt{host}$
    $State_B \leftarrow$ 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

Fig. 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: `balance`$_P \leftarrow$ `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: `balance`$_P \leftarrow$ `balance`$_P + x$
20: // if $\bar{P}$ honest, this state transition happens simultaneously with l. 15
21: $State_P \leftarrow$ OPEN
22: **end if**

Fig. 3.

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

1: On input (FUND ME, x, $\ldots$) by ITI $R \in \{\mathcal{F}_{\mathrm{Chan}}, \mathrm{LN}\}$ when $State_P =$ OPEN:
2:     store x
3:     $State_P \leftarrow$ PENDING FUND

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

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

12: When $State_P =$ OPEN:
13:     **if** we intercept the command "define new VIRT ITI `host`" by $\mathcal{A}$, routed through P **then**
14:         store `host`
15:         $State_P \leftarrow$ TENTATIVE HELP FUND
16:         continue executing $\mathcal{A}$'s command
17:     **end if**

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

20: When $State_P =$ SYNC FUND:
21:     **if** $State_{\bar{P}} \in \{\mathrm{IGNORED}, \mathrm{SYNC\ HELP\ FUND}\}$ **then**
22:         $\mathtt{balance}_P \leftarrow \mathtt{balance}_P - x$
23:         $\mathtt{host}_P \leftarrow \mathtt{host}$
24:         // if $\bar{P}$ honest, this state transition happens simultaneously with l. 31
25:         $State_P \leftarrow$ OPEN
26:     **end if**

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

Fig. 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 (CLOSE, P) by $\mathcal{A}$ when $State \notin \{$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
    $\texttt{coins}_P \leftarrow$ sum of coins exclusively spendable by pk_P in Σ
    if $\texttt{coins}_P \geq \texttt{balance}_P$ **then**
        $State_P \leftarrow$ CLOSED
    else // balance security is broken
        halt
    end if

Fig. 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.

Fig. 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, 22): 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 $\mathcal{F}_{\text{Chan}}$

Fig. 7.

Process LN – init

// When not specified, input comes from and output goes to $\mathcal{E}$.
// The ITI knows whether it is *Alice* (funder) or *Bob* (fundee). The activated party is P and the counterparty is $\bar{P}$.
On every activation, before handling the message:
  if `last_poll` $\neq \perp$ **then** // channel has opened
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    if `last_poll` $+ t < |\Sigma|$ **then**
      `negligent` $\leftarrow$ True
    end if
  end if

On (INIT, $pk_{P,\text{out}}$):
  ensure $State = \perp$
  $State \leftarrow$ INIT
  store $pk_{P,\text{out}}$
  $(c_A, c_B, \texttt{locked}_A, \texttt{locked}_B) \leftarrow (0, 0, 0, 0)$
  $(\texttt{paid\_out}, \texttt{paid\_in}) \leftarrow (\emptyset, \emptyset)$
  `negligent` $\leftarrow$ False
  `last_poll` $\leftarrow \perp$
  output (INIT OK)

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

On (BALANCE):
  ensure $State^P \in \{\text{OPEN}, \text{CLOSED}\}$
  output (BALANCE, $c_A, c_B, \texttt{locked}_A, \texttt{locked}_B$)

Fig. 8.

Process LN – methods used by VIRT

REVOKEPREVIOUS():
  ensure $State \in$ 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**
    $State \leftarrow$ WAITING FOR INBOUND REVOCATION
  else // $State =$ WAITING FOR OUTBOUND REVOCATION
    $i \leftarrow i + 1$
    $State \leftarrow$ WAITING FOR HOSTS READY
  end if
  $\texttt{host}_P \leftarrow \texttt{host}'_P$ // forget old host, use new host instead
  $\texttt{layer} \leftarrow \texttt{layer} + 1$
  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 $\text{VERIFY}(R_{P,i}, \text{sig}_{\bar{P},R,i}, pk_{\bar{P},R}) = \text{True}$
  if $State =$ WAITING FOR REVOCATION **then**
    $State \leftarrow$ WAITING FOR OUTBOUND REVOCATION
  else // $State =$ WAITING FOR INBOUND REVOCATION
    $i \leftarrow i + 1$
    $State \leftarrow$ WAITING FOR HOSTS READY
  end if
  return (OK)

NEGLIGENT():
  $\texttt{negligent} \leftarrow \text{True}$
  return (OK)

Fig. 9.

Process LN.EXCHANGEOPENKEYS()

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

State $\leftarrow$ 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` $\leftarrow 0$

    State $\leftarrow$ WAITING FOR COMM SIG

else // opening virtual channel

    State $\leftarrow$ 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 $\leftarrow$ OPENING KEYS OK

Fig. 10.

Process LN.PREPAREBASE()

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

    $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}

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

    `layer` $\leftarrow 0$

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, `host`$_P$, `funder_layer`) // ignore any other message

    `layer` $\leftarrow$ `funder_layer`

end if

Fig. 11.

Process LN.EXCHANGEOPENSIGS()

// $s = (2 + \lceil \texttt{maxTime}_{\texttt{window}} + \frac{\texttt{Delay}}{2} / \texttt{minTime}_{\texttt{window}} \rceil) \texttt{windowSize}$, where $\texttt{maxTime}_{\texttt{window}}$, $\texttt{Delay}$, $\texttt{minTime}_{\texttt{window}}$ and $\texttt{windowSize}$ are defined in Proposition ?? TODO: recheck and include proposition
$C_{A,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, 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: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, 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: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}
end if
$C_{B,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, pk_{A,\text{out}})$, $(0, (pk_{B,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}
ensure $\text{VERIFY}(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F}) = \text{True}$
$C_{A,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, 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 $\text{VERIFY}(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F}) = \text{True}$

Fig. 12.

Process LN.COMMITBASE()

$\text{sig}_F \leftarrow \text{SIGN}(F, sk_{A,\text{chain}})$
send (OPEN, $c, pk_{A,\text{out}}, pk_{B,\text{out}}, F, \text{sig}_F, Alice, Bob$) to $\mathcal{A}$
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

Fig. 13.

Process LN – external open messages for *Bob*

On input (CHECK FUNDING):
  ensure *State* = WAITING TO CHECK FUNDING
  input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
  if $F \in \Sigma$ **then**
    State ← OPEN
    reply (OPEN OK)
  end if

On output (FUNDED, `host`$_P$, `funder_layer`) by `hops`[-1].`right`:
  ensure *State* = WAITING FOR FUNDED
  store `host`$_P$ // we will talk directly to `host`$_P$
  `layer` ← `funder_layer`
  State ← WAITING FOR COMM SIG
  reply (FUND ACK)

On output (CHECK KEYS, (pk_1, pk_2)) by `hops`[-1].`right`:
  ensure *State* = WAITING FOR CHECK KEYS
  ensure $pk_1 = pk_{A,F} \wedge pk_2 = pk_{B,F}$
  State ← WAITING FOR FUDNED
  reply (KEYS OK)

Fig. 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`)

Fig. 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_{guest}
$\text{sig}_{P,C,i+1} \leftarrow \text{SIGN}(C_{\bar{P},i+1})$ // 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_{guest}
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_{guest}
$\text{sig}_{\bar{P},C,i+1} \leftarrow \text{SIGN}(C_{P,i+1}, sk'_{\bar{P},F})$ // 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_{guest}
ensure VERIFY$(C_{P,i+1}, \text{sig}_{\bar{P},C,i+1}, pk'_{\bar{P},F}) =$ True

Fig. 16.

Process LN – virtualise start and end

On input (FUND ME, c_{guest}, `fundee`, `hops`, $pk_{A,V}$, $pk_{B,V}$) by `funder`:
 ensure $State$ = OPEN
 ensure $c_P -$ `locked`$_P \geq c_{\text{guest}}$
 $State \leftarrow$ VIRTUALISING
 $(sk'_{P,F}, pk'_{P,F}) \leftarrow$ KEYGEN()
 define new VIRT ITI `host`$'_P$
 send (VIRTUALISING, `host`$'_P$, $pk'_{P,F}$, `hops`, `fundee`, c_{guest}) to $\bar{P}$ and expect reply (VIRTUALISING ACK, `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()
 $State \leftarrow$ WAITING FOR REVOCATION
 input (HOST ME, `funder`, `fundee`, `host`$'_{\bar{P}}$, `host`$_P$, c_{guest}, $pk_{A,V}$, $pk_{B,V}$, $(sk'_{P,F}$, $pk'_{P,F})$, $(sk_{P,F}$, $pk_{P,F})$, $pk_{\bar{P},F}$, $pk'_{\bar{P},F})$ to `host`$'_P$

On output (HOSTS READY) by `host`$_P$: // `host`$_P$ is the new host, renamed in Fig. 9, l. 12
 ensure $State$ = WAITING FOR HOSTS READY
 $State \leftarrow$ 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, `host`$_P$, `layer`) 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
  $c_P \leftarrow c_P - c_{\text{guest}}$
  output (FUNDED, `host`$_P$, `layer`) to `funder` // do not expect reply by `funder`
 end if
 reply (HOST ACK)

On output (SIGN TXS, TXs) by `host`$'_P$:
 sigs $\leftarrow \emptyset$
 for TX in TXs **do**
  add SIGN(TX, $sk_{P,F}$, ANYPREVOUT) to sigs
 end for
 reply (TXS SIGNED, sigs)

Fig. 17.

Process LN – virtualise hops

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

On input (VIRTUALISING, $\texttt{host\_sibling}$, $(sk'_{P,F}, pk'_{P,F})$, $pk_{\text{sib},\bar{P},F}$, $\texttt{hops}$, $\texttt{fundee}$, c_{guest}, $c_{\text{sib,rem}}$, $_{\text{sib}}$) by $\texttt{sibling}$:
    ensure $State =$ OPEN
    ensure $c_P - \texttt{locked}_P \geq c$
    ensure $c_{\text{sib,rem}} \geq c_P \wedge c_{\bar{P}} \geq c_{\text{sib}}$ // avoid value loss by griefing attack: one counterparty closes with old version, the other stays idle forever
    $State \leftarrow$ VIRTUALISING
    $\texttt{locked}_P \leftarrow \texttt{locked}_P + c$
    define new VIRT ITI $\texttt{host}'_P$
    send (VIRTUALISING, $\texttt{host}'_P$, $pk'_{P,F}$, $\texttt{hops}$, $\texttt{fundee}$, c_{guest}) to $\texttt{hops}$[0].$\texttt{right}$ 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()
    input (INIT, $\texttt{host}_P$, $\texttt{host}'_{\bar{P}}$, $\texttt{host\_sibling}$, $(sk'_{P,F}, pk'_{P,F})$, $pk'_{\bar{P},F}$, $pk_{\text{sib},\bar{P},F}$, $(sk_{P,F}, pk_{P,F})$, $pk_{\bar{P},F}$, c_{guest}) to $\texttt{host}'_P$ and expect reply (HOST INIT OK)
    $State \leftarrow$ WAITING FOR REVOCATION
    output (VIRTUALISING ACK, $\texttt{host}'_P$, $pk'_{\bar{P},F}$) to $\texttt{sibling}$

Fig. 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
$\mathrm{sig}_{P,C,i+1} \leftarrow \textsc{sign}(C_{\bar{P},i+1}, sk_{P,F})$ // kept by $\bar{P}$
$\mathit{State} \leftarrow$ WAITING FOR COMMITMENT SIGNED
send (PAY, x, $\mathrm{sig}_{P,C,i+1}$) to $\bar{P}$
// P refers to payer and $\bar{P}$ to payee both in local and remote code
ensure $\mathit{State} =$ WAITING TO GET PAID $\wedge\, x = y$
if $\mathtt{host}_{\bar{P}} \neq \mathcal{G}_{\mathrm{Ledger}} \wedge \bar{P}$ 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
end if
$C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with x coins moved from P's to $\bar{P}$'s output
ensure $\textsc{verify}(C_{\bar{P},i+1}, \mathrm{sig}_{P,C,i+1}, pk_{P,F}) =$ True
$C_{P,i+1} \leftarrow C_{P,i}$ with x coins moved from P's to $\bar{P}$'s output
$\mathrm{sig}_{\bar{P},C,i+1} \leftarrow \textsc{sign}(C_{P,i+1}, sk_{\bar{P},F})$ // kept by P
$R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: $(c_{\bar{P}}, pk_{P,\mathrm{out}})$}
$\mathrm{sig}_{\bar{P},R,i} \leftarrow \textsc{sign}(R_{P,i}, sk_{\bar{P},R})$
$\mathit{State} \leftarrow$ WAITING FOR PAY REVOCATION
reply (COMMITMENT SIGNED, $\mathrm{sig}_{\bar{P},C,i+1}$, $\mathrm{sig}_{\bar{P},R,i}$)
ensure $\mathit{State} =$ WAITING FOR COMMITMENT SIGNED
$C_{P,i+1} \leftarrow C_{P,i}$ with x coins moved from P's to $\bar{P}$'s output

Fig. 19.

Process LN.REVOCATIONSTRIP()

ensure VERIFY($C_{P,i+1}$, $\text{sig}_{\bar{P},C,i+1}$, $pk_{\bar{P},F}$) = True
$R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: ($c_{\bar{P}}$, $pk_{P,\text{out}}$)}
ensure VERIFY($R_{P,i}$, $\text{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
$R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: (c_P, $pk_{\bar{P},\text{out}}$)}
$\text{sig}_{P,R,i} \leftarrow$ 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 `host`$_P \neq \mathcal{G}_{\text{Ledger}} \wedge$ we have a `host_sibling` **then** // we are intermediary channel
    input (NEW BALANCE, c_P, $c_{\bar{P}}$) to `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, $\text{sig}_{P,R,i}$) 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, $pk_{\bar{P},\text{out}}$)}
ensure VERIFY($R_{\bar{P},i}$, $\text{sig}_{P,R,i}$, $pk_{P,R}$) = 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 `host`$_P \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$ has a `host_sibling` **then** // we are intermediary channel
    input (NEW BALANCE, $c_{\bar{P}}$, c_P) to `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

Fig. 20.

Process LN – On (PAY, x):

ensure $\mathit{State} = \text{OPEN} \wedge c_P \geq x$
if `host`$_P \neq \mathcal{G}_{\text{Ledger}} \wedge P$ has a `host_sibling` **then** // we are intermediary channel
    ensure $c_{\text{sib,rem}} \geq c_P - x \wedge c_{\bar{P}} + x \geq c_{\text{sib}}$ // 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

Fig. 21.

Process LN – On (GET PAID, y):

ensure $\mathit{State} = \text{OPEN} \wedge c_{\bar{P}} \geq x$
store y
$\mathit{State} \leftarrow$ WAITING TO GET PAID

Fig. 22.

Process LN – On (CHECK FOR LATERAL CLOSE):

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

Fig. 23.

Process LN – On (CHECK CHAIN FOR CLOSED):

ensure $\mathit{State} \notin \{\bot, \text{INIT}, \text{TOPPED UP}\}$ // channel open
// even virtual channels check $\mathcal{G}_{\text{Ledger}}$ directly. This is intentional
input (READ) to $\mathcal{G}_{\text{Ledger}}$ 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
    $\mathit{State} \leftarrow$ CLOSING
    LN.SUBMITANDCHECKREVOCATION(j)
    $\mathit{State} \leftarrow$ CLOSED
    output (CLOSED)
else if $C_{P,j} \in \Sigma \wedge C_{\bar{P},j} \in \Sigma$ **then**
    $\mathit{State} \leftarrow$ CLOSED
    output (CLOSED)
end if

Fig. 24.

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

Fig. 25.

Process LN – On (CLOSE):

ensure $State \notin \{\bot, \text{INIT}, \text{TOPPED UP}, \text{CLOSED}, \text{BASE PUNISHED}\}$ // 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 inputs (CHECK CHAIN FOR CLOSING) 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 `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

Fig. 26.

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

$State \leftarrow$ BASE PUNISHED

Fig. 27.

Process VIRT

On every activation, before handling the message:
  if `last_poll` $\neq \perp$ **then** // virtual layer is ready
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    if `last_poll` $+ t < |\Sigma|$ **then**
      for $P \in \{$`guest`, `funder`, `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 is redundant
On input (INIT, $\texttt{host}_P$, $\bar{P}$, `sibling`, `fundee`, $(sk_{\text{loc,virt}}, pk_{\text{loc,virt}})$, $pk_{\text{rem,virt}}$, $pk_{\text{sib,rem,virt}}$, $(sk_{\text{loc},F}, pk_{\text{loc},F})$, $pk_{\text{rem},F}$, c_{guest}) by `guest`:
  store message contents and `guest` // `sibling`, $pk_{\text{sib},\bar{P},F}$ are missing for edge nodes, `fundee` is present only in last node
  `last_poll` $\leftarrow \perp$
  output (HOST INIT OK) to `guest`

On input (HOST ME, `funder`, `fundee`, $\bar{P}$, $\texttt{host}_P$, c_{guest}, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, $(sk_{\text{loc,virt}}, pk_{\text{loc,virt}})$, $(sk_{\text{loc},F}, pk_{\text{loc},F})$, $pk_{\text{rem},F}$, $pk_{\text{rem,virt}}$) by `guest`:
  `last_poll` $\leftarrow \perp$
  ensure VIRT.CIRCULATEKEYSANDCOINS() returns (OK)
  ensure VIRT.CIRCULATEVIRTUALSIGS() returns (OK)
  ensure VIRT.CIRCULATEFUNDINGSIGS() returns (OK)
  ensure VIRT.CIRCULATEREVOCATIONS() returns (OK)
  output (HOSTS READY) to `guest`

Fig. 28.

Process VIRT.CIRCULATEKEYSANDCOINS(`left_data`):

if `left_data` is given as argument **then** // we are not `host_funder`
  if we have a `sibling` **then** // we are not `host_fundee`
    input (KEYS AND COINS FORWARD, (`left_data`, $(sk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}})$, $(sk_{\mathrm{loc},F}, pk_{\mathrm{loc},F})$, $pk_{\mathrm{rem},F}$, c_P, $c_{\bar{P}}$) to `sibling`
    store input as `left_data`
    parse `left_data` as `far_left_data`, $(sk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}})$, $(sk_{\mathrm{sib},F}, pk_{\mathrm{sib},F})$, $pk_{\mathrm{sib,rem},F}$, c_{sib}, $c_{\mathrm{sib,rem}}$ // remove parentheses as necessary
    call VIRT.CIRCULATEKEYSANDCOINS(`left_data`) of $\bar{P}$ and assign returned value to `right_data`
    parse `right_data` as `far_right_data`, $pk_{\mathrm{rem,virt}}$
    output (KEYS AND COINS BACK, `right_data`, $(sk_{\mathrm{loc},F}, pk_{\mathrm{loc},F})$, $pk_{\mathrm{rem},F}$, c_P, $c_{\bar{P}}$)
    store output as `right_data`
    parse `right_data` as `far_right_data`, $(sk_{\mathrm{sib},F}, pk_{\mathrm{sib},F})$, $pk_{\mathrm{sib,rem},F}$, c_{sib}, $c_{\mathrm{sib,rem}}$
    return (`right_data`, $pk_{\mathrm{loc,virt}}$)
  else // we are `host_fundee`
    extract $(pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}})$ from `left_data`
    output (CHECK KEYS, $(pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}})$) to `fundee` and expect reply (KEYS OK)
    return $pk_{\mathrm{loc,virt}}$
  end if
else // we are `host_funder`
  call VIRT.CIRCULATEKEYSANDCOINS($pk_{\mathrm{loc,virt}}$, $(pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}})$) of $\bar{P}$ and assign returned value to `right_data`
  return (OK)
end if

Fig. 29.

Process VIRT

$\textsc{getMidTXs}(c_{\text{guest}}, c_{\text{loc}}, c_{\text{rem}}, c_{\text{sib}}, c_{\text{sibRem}}, pk_{\text{left,fund}}, pk_{\text{loc,fund}}, pk_{\text{sib,fund}}, pk_{\text{right,fund}}, pk_{\text{left,virt}}, pk_{\text{loc,virt}}, pk_{\text{sib,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}, pk_{\text{loc,out}}, \{pk_{\text{sec},i}\}_{i \in 1 \ldots n})$:

  ensure $c_{\text{sibRem}} \geq c_{\text{guest}} \wedge c_{\text{loc}} \geq c_{\text{guest}}$

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

  $\texttt{left\_fund} \leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{loc,fund}}\}$

  $\texttt{right\_fund} \leftarrow 2/\{pk_{\text{sib,fund}}, pk_{\text{right,fund}}\}$

  $\texttt{left\_virt} \leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}\}$

  $\texttt{left\_virt\_checked} \leftarrow 4/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  $\texttt{right\_virt} \leftarrow 2/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}\}$

  $\texttt{right\_virt\_checked} \leftarrow 4/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  $\texttt{left\_out\_checked} \leftarrow (\texttt{2} \wedge \texttt{left\_virt\_checked}) \vee (\texttt{3} \wedge \texttt{left\_virt} + (t+s))$

  $\texttt{right\_out} \leftarrow (\texttt{1} \wedge \texttt{right\_virt}) \vee (\texttt{3} \wedge \texttt{right\_virt} + (t+s))$

  $\texttt{right\_out\_checked} \leftarrow (\texttt{1} \wedge \texttt{right\_virt\_checked}) \vee (\texttt{3} \wedge \texttt{right\_virt} + (t+s))$

  $\texttt{guest\_all} \leftarrow \texttt{5} \wedge n/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1\ldots n}\}\}$

  $\texttt{guest\_out} \leftarrow \texttt{4} \wedge 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  $\texttt{guest} \leftarrow (\texttt{guest\_out} + (t+s)) \vee \texttt{guest\_all}$

  $\text{TX}_{\text{none}} \leftarrow$ TX {inputs: $((c_{\text{left}}, \texttt{left\_fund}), (c_{\text{right}}, \texttt{right\_fund}))$, outputs: $((c_{\text{left}} - c_{\text{guest}}, \texttt{left\_out\_checked}), (c_{\text{right}} - c_{\text{guest}}, \texttt{right\_out\_checked}), (c_{\text{guest}}, pk_{\text{loc,out}}), (c_{\text{guest}}, \texttt{guest}))$}

  $\text{TX}_{\text{left}} \leftarrow$ TX {inputs: $((c_{\text{left}} - c_{\text{guest}}, \texttt{1} \wedge \texttt{left\_virt\_checked}), (c_{\text{right}}, \texttt{right\_fund}))$, outputs: $((c_{\text{left}} - c_{\text{guest}}, \texttt{3} \wedge \texttt{left\_virt}), (c_{\text{right}} - c_{\text{guest}}, \texttt{right\_out\_checked}), (c_{\text{guest}}, pk_{\text{loc,out}}))$}

  $\text{TX}_{\text{right}} \leftarrow$ TX {inputs: $((c_{\text{left}}, \texttt{left\_fund}), (c_{\text{right}} - c_{\text{guest}}, \texttt{2} \wedge \texttt{right\_virt\_checked}), (c_{\text{guest}}, \texttt{guest\_all}))$, outputs: $((c_{\text{left}} - c_{\text{guest}}, \texttt{left\_out\_checked}), (c_{\text{right}} - c_{\text{guest}}, \texttt{3} \wedge \texttt{right\_virt}), (c_{\text{guest}}, pk_{\text{loc,out}}), (c_{\text{guest}}, \texttt{guest}))$}

  $\text{TX}_{\text{both}} \leftarrow$ TX {inputs: $((c_{\text{left}} - c_{\text{guest}}, \texttt{1} \wedge \texttt{left\_virt\_checked}), (c_{\text{right}} - c_{\text{guest}}, \texttt{2} \wedge \texttt{right\_virt\_checked}), (c_{\text{guest}}, \texttt{guest\_all}))$, outputs: $((c_{\text{left}} - c_{\text{guest}}, \texttt{3} \wedge \texttt{left\_virt}), (c_{\text{right}} - c_{\text{guest}}, \texttt{3} \wedge \texttt{right\_virt}), (c_{\text{guest}}, pk_{\text{loc,out}})$}

  return $(\text{TX}_{\text{none}}, \text{TX}_{\text{left}}, \text{TX}_{\text{right}}, \text{TX}_{\text{both}})$

Fig. 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.

GETEDGETXS(c_{guest}, c_{left}, c_{right}, $pk_{\text{left,fund}}$, $pk_{\text{right,fund}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$, `is_funder`):

  ensure $c_{\text{left}} \geq c_{\text{guest}}$

  $c_{\text{tot}} \leftarrow c_{\text{left}} + c_{\text{right}}$

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

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

  `virt_checked` $\leftarrow 4/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  if `is_funder` = True **then**

    `out` $\leftarrow$ (`1` $\wedge$ `virt_checked`) $\vee$ (`3` $\wedge$ `virt` $+ (t+s)$)

  else // TXs belong to `fundee`

    `out` $\leftarrow$ (`2` $\wedge$ `virt_checked`) $\vee$ (`3` $\wedge$ `virt` $+ (t+s)$)

  end if

  `guest_all` $\leftarrow$ `5` $\wedge\ n/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1 \dots n}\}\}$

  `guest_out` $\leftarrow$ `4` $\wedge\ 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  `guest` $\leftarrow$ (`guest_out` $+ (t+s)$) $\vee$ `guest_all`

  $\text{TX}_{\text{base}} \leftarrow$ TX {input: (c_{tot}, `fund`), outputs: (($c_{\text{tot}} - c_{\text{guest}}$, `out`), ($c_{\text{guest}}$, `guest`))}

  return TX_{base}

Fig. 31.

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

parse input as $\mathrm{sigs}_{\mathrm{byLeft}}$
$(\mathrm{TX}_{\mathrm{loc,none}}, \mathrm{TX}_{\mathrm{loc,left}}, \mathrm{TX}_{\mathrm{loc,right}}, \mathrm{TX}_{\mathrm{loc,both}}) \leftarrow \textsc{virt.getMidTXs}(c_{\mathrm{guest}}, c_P, c_{\bar{P}}, c_{\mathrm{sib}}, c_{\mathrm{sib,rem}}, pk_{\mathrm{sib,rem},F}, pk_{\mathrm{sib},F}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{sib,rem,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, pk_{\mathrm{loc,virt}}, \{pk_{\mathrm{sec},i}\}_{i\in 1\ldots n})$
store all signatures in $\mathrm{sigs}_{\mathrm{byLeft}}$ that sign any of $\mathrm{TX}_{\mathrm{loc,none}}$, $\mathrm{TX}_{\mathrm{loc,left}}$, $\mathrm{TX}_{\mathrm{loc,right}}$, $\mathrm{TX}_{\mathrm{loc,both}}$ and remove these signatures from $\mathrm{sigs}_{\mathrm{byLeft}}$
ensure that the stored signatures contain one valid signature for $\mathrm{TX}_{\mathrm{loc,right}}$ and $\mathrm{TX}_{\mathrm{loc,both}}$ which sign the `guest_all` input by each one of the previous $j-1$ hops
ensure that there are exactly 4 more valid signatures in the stored signatures, which sign the $1 \wedge$ `left_virt_checked` inputs of $\mathrm{TX}_{\mathrm{loc,left}}$ and $\mathrm{TX}_{\mathrm{loc,both}}$ with $pk_{\mathrm{sib,rem,virt}}$ and $pk_{\mathrm{left,guest}}$
$\mathrm{sigs}_{\mathrm{toRight}} \leftarrow \mathrm{sigs}_{\mathrm{byLeft}}$
for each hop apart from the first, the last and ours ($i \in [2, \ldots, n-1] \setminus \{j\}$) **do** // j is our hop number, hop data encoded in `left_data` and `right_data`
    extract data needed for $\textsc{getMidTXs}()$ from `left_data` (if $i < j$) or `right_data` (if $i > j$) and assign it to $\mathtt{data}_i$ and $\{pk_{\mathrm{sec},i}\}_{i\in 1\ldots n}$ // P and `comm_keys` are missing, that is OK. $\{pk_{\mathrm{sec},i}\}_{i\in 1\ldots n}$ contains each party's $pk_{i,\mathrm{virt}}$
    $(\mathrm{TX}_{i,\mathrm{none}}, \mathrm{TX}_{i,\mathrm{left}}, \mathrm{TX}_{i,\mathrm{right}}, \mathrm{TX}_{i,\mathrm{both}}) \leftarrow \textsc{virt.getMidTXs}(\mathtt{data}_i, \{pk_{\mathrm{sec},i}\}_{i\in 1\ldots n})$
    add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{right}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ and $\textsc{sign}(\mathrm{TX}_{i,\mathrm{both}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$ if $i < j$, or $\mathrm{sigs}_{\mathrm{toRight}}$ if $i > j$ // if i-th hop is adjacent, 2 signatures will be produced by each $\textsc{sign}()$ invocation: one for the `guest_all` and one for the $2 \wedge$ `right_virt_checked` input
    if $i - j = 1$ **then** // hop is our next
        add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{left}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toRight}}$
    else if $j - i = 1$ **then** // hop is our previous
        add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{left}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$
    end if
end for
if `right_data` does not contain data from a second-next hop **then** // next hop is `host_fundee`
    $\mathrm{TX}_{\mathrm{next,none}} \leftarrow \textsc{virt.getEdgeTXs}(c_{\mathrm{guest}}, c_P, c_{\bar{P}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, \mathrm{False})$
end if
call $\bar{P}.\textsc{circulateVirtualSigs}(\mathrm{sigs}_{\mathrm{toRight}})$ and assign returned value to $\mathrm{sigs}_{\mathrm{byRight}}$
store all signatures in $\mathrm{sigs}_{\mathrm{byRight}}$ that sign any of $\mathrm{TX}_{\mathrm{loc,none}}$, $\mathrm{TX}_{\mathrm{loc,left}}$, $\mathrm{TX}_{\mathrm{loc,right}}$, $\mathrm{TX}_{\mathrm{loc,both}}$ and remove these signatures from $\mathrm{sigs}_{\mathrm{byRight}}$
ensure that the stored signatures contain one valid signature for $\mathrm{TX}_{\mathrm{loc,right}}$ and $\mathrm{TX}_{\mathrm{loc,both}}$ which sign the `guest_all` input by each one of the next $n - j$ hops
ensure that there are exactly 4 more valid signatures in the stored signatures, which sign the $2 \wedge$ `right_virt_checked` inputs of $\mathrm{TX}_{\mathrm{loc,right}}$ and $\mathrm{TX}_{\mathrm{loc,both}}$ with $pk_{\mathrm{rem,virt}}$ and $pk_{\mathrm{right,guest}}$
output $(\textsc{virtualSigsBack}, \mathrm{sigs}_{\mathrm{toLeft}}, \mathrm{sigs}_{\mathrm{byRight}})$

Fig. 32.

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

$(\mathrm{TX}_{\mathrm{loc,none}}, \mathrm{TX}_{\mathrm{loc,left}}, \mathrm{TX}_{\mathrm{loc,right}}, \mathrm{TX}_{\mathrm{loc,both}}) \leftarrow \textsc{virt.getMidTXs}(c_{\mathrm{guest}}, c_P, c_{\bar{P}}, c_{\mathrm{sib}}, c_{\mathrm{sib,rem}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{sib},F}, pk_{\mathrm{sib,rem}F}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{sib,rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, pk_{\mathrm{loc,virt}}, \{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n})$
// not verifying our signatures in $\mathrm{sigs}_{\mathrm{byLeft}}$, our (trusted) `sibling` will do that
input ($\textsc{virtual sigs forward}$, $\mathrm{sigs}_{\mathrm{byLeft}}$) to `sibling`
$\textsc{virt.siblingSigs}()$
$\mathrm{sigs}_{\mathrm{toLeft}} \leftarrow \mathrm{sigs}_{\mathrm{byRight}} + \mathrm{sigs}_{\mathrm{toLeft}}$
if `left_data` does not contain data from a second-previous hop **then** // previous hop is `host_funder`
    $\mathrm{TX}_{\mathrm{prev,none}} \leftarrow \textsc{virt.getEdgeTXs}(c_{\mathrm{guest}}, c_{\bar{P}}, c_P, pk_{\mathrm{rem},F}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, \mathrm{True})$
end if
return $\mathrm{sigs}_{\mathrm{toLeft}}$

Fig. 33.

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

$\mathrm{TX}_{\mathrm{loc,none}} \leftarrow \textsc{virt.getEdgeTXs}(c_{\mathrm{guest}}, c_P, c_{\bar{P}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, \mathrm{False})$
for each hop apart from the first and ours ($i \in [2, \dots, n-1]$) **do** // hop data encoded in `left_data`
    extract data needed for $\textsc{getMidTXs}()$ from `left_data` and assign it to $\mathtt{data}_i$ and $\{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n}$ // $\{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n}$ contains each party's $pk_{i,\mathrm{virt}}$
    $(\mathrm{TX}_{i,\mathrm{none}}, \mathrm{TX}_{i,\mathrm{left}}, \mathrm{TX}_{i,\mathrm{right}}, \mathrm{TX}_{i,\mathrm{both}}) \leftarrow \textsc{virt.getMidTXs}(\mathtt{data}_i, \{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n})$
    add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{right}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ and $\textsc{sign}(\mathrm{TX}_{i,\mathrm{both}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$ // if i-th hop is adjacent, 2 signatures will be produced by each $\textsc{sign}()$ invocation: one for the `guest_all` and one for the $2 \wedge$ `right_virt_checked` input
    output ($\textsc{sign TXs}$, $\mathrm{TX}_{i,\mathrm{left}}$, $\mathrm{TX}_{i,\mathrm{right}}$, $\mathrm{TX}_{i,\mathrm{both}}$) to `fundee` and expect reply ($\textsc{TXs signed}$, $\mathrm{sigs}_{\mathrm{guest}}$)
    add $\mathrm{sigs}_{\mathrm{guest}}$ to $\mathrm{sigs}_{\mathrm{toLeft}}$
    if $i = n - 1$ **then** // hop is our previous
        add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{left}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$
    end if
end for
return $\mathrm{sigs}_{\mathrm{toLeft}}$

Fig. 34.

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

for each hop apart from the last and ours ($i \in [2, \ldots, n-1]$) **do** // hop data encoded in `right_data`
  extract data needed for $\textsc{getMidTXs}()$ from `right_data` and assign it to $\texttt{data}_i$ and $\{pk_{\mathrm{sec},i}\}_{i \in 1 \ldots n}$ // $\{pk_{\mathrm{sec},i}\}_{i \in 1 \ldots n}$ contains each party's $pk_{i,\mathrm{virt}}$
  $(\mathrm{TX}_{i,\mathrm{none}}, \mathrm{TX}_{i,\mathrm{left}}, \mathrm{TX}_{i,\mathrm{right}}, \mathrm{TX}_{i,\mathrm{both}}) \leftarrow \textsc{virt.getMidTXs}(\texttt{data}_i, \{pk_{\mathrm{sec},i}\}_{i \in 1 \ldots n})$
  add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{right}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ and $\textsc{sign}(\mathrm{TX}_{i,\mathrm{both}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toRight}}$ // if i-th hop is adjacent, 2 signatures will be produced by each $\textsc{sign}()$ invocation: one for the `guest_all` and one for the $2 \wedge$ `right_virt_checked` input
  output ($\textsc{sign TXs}$, $\mathrm{TX}_{i,\mathrm{left}}$, $\mathrm{TX}_{i,\mathrm{right}}$, $\mathrm{TX}_{i,\mathrm{both}}$) to `fundee` and expect reply ($\textsc{TXs signed}$, $\mathrm{sigs}_{\mathrm{guest}}$)
  add $\mathrm{sigs}_{\mathrm{guest}}$ to $\mathrm{sigs}_{\mathrm{toRight}}$
  if $i = 2$ **then** // hop is our next
    add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{left}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toRight}}$
  end if
end for
call $\textsc{virt.circulateVirtualSigs}(\mathrm{sigs}_{\mathrm{toRight}})$ of P and assign output to $\mathrm{sigs}_{\mathrm{byRight}}$
$\mathrm{TX}_{\mathrm{loc,none}} \leftarrow \textsc{virt.getEdgeTXs}(c_{\mathrm{guest}}, c_P, c_{\bar{P}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, \mathrm{True})$
return ($\textsc{ok}$)

Fig. 35.

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

if $\mathrm{sigs}_{\mathrm{byLeft}}$ is given as argument **then** // we are not `host_funder`
  if we have a `sibling` **then** // we are not `host_fundee`
    return $\textsc{virt.intermediarySigs}()$
  else // we are `host_fundee`
    return $\textsc{virt.hostFundeeSigs}()$
  end if
else // we are `host_funder`
  return $\textsc{virt.hostFunderSigs}()$
end if

Fig. 36.

Process $\textsc{virt.circulateFundingSigs}(\text{sig}_{\text{loc,none}}, \text{sig}_{\text{loc,right}})$

if $\text{sig}_{\text{loc,none}}$ is given as argument **then** // we are not `host_funder`
  ensure $\textsc{verify}(\text{TX}_{\text{loc,none}}, \text{sig}_{\text{loc,none}}, pk_{\text{prev},F}) = \text{True}$ // $pk_{\text{prev},F}$ found in `left_data`
  $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
  if $\text{sig}_{\text{loc,right}}$ is given as argument **then** // we are not `host_fundee`
    ensure $\textsc{verify}(\text{TX}_{\text{loc,right}}, \text{sig}_{\text{loc,right}}, pk_{\text{prev},F}) = \text{True}$
    $\text{sigs}_{\text{loc,right}} \leftarrow \{\text{sig}_{\text{loc,right}}\}$
    input ($\textsc{virtual base sig forward}$, $\text{sig}_{\text{loc,none}}$) to **`sibling`** // `sibling` needs $\text{sig}_{\text{loc,none}}$ for closing
    $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
    $\text{sig}_{\text{next,none}} \leftarrow \textsc{sign}(\text{TX}_{\text{next,none}}, sk_{\text{loc},F})$
    `args` $\leftarrow \{\text{sig}_{\text{next,none}}\}$
    if `right_data` contains data from a second-next hop **then** // next hop is not `host_fundee`
      $\text{sig}_{\text{next,right}} \leftarrow \textsc{sign}(\text{TX}_{\text{next,right}}, sk_{\text{loc},F})$
      add $\text{sig}_{\text{next,right}}$ to **`args`**
    end if
    call $\textsc{virt.circulateFundingSigs}$(**`args`**) of $\bar{P}$ and assign returned values to $(\text{sig}_{\text{loc,none}}, \text{sig}_{\text{loc,left}})$
    ensure $\textsc{verify}(\text{TX}_{\text{loc,none}}, \text{sig}_{\text{loc,none}}, pk_{\text{next},F}) = \text{True}$ // $pk_{\text{next},F}$, found in `right_data`
    ensure $\textsc{verify}(\text{TX}_{\text{loc,left}}, \text{sig}_{\text{loc,left}}, pk_{\text{next},F}) = \text{True}$
    add $\text{sig}_{\text{loc,none}}$ to $\text{sigs}_{\text{loc,none}}$; $\text{sigs}_{\text{loc,left}} \leftarrow \{\text{sig}_{\text{loc,left}}\}$
    output ($\textsc{virtual base sig back}$, $\text{sig}_{\text{loc,none}}$) // sibling needs $\text{sig}_{\text{loc,none}}$ for closing
    add $\text{sig}_{\text{loc,none}}$ to $\text{sigs}_{\text{loc,none}}$
  end if
  $\text{sig}_{\text{prev,none}} \leftarrow \textsc{sign}(\text{TX}_{\text{prev,none}}, sk_{\text{loc},F})$
  if `left_data` contains data from a second-previous hop **then** // previous hop not is `host_funder`
    $\text{sig}_{\text{prev,left}} \leftarrow \textsc{sign}(\text{TX}_{\text{prev,left}}, sk_{\text{loc},F})$
    return $(\text{sig}_{\text{prev,none}}, \text{sig}_{\text{prev,left}})$
  else // previous hop is `host_funder`
    return $\text{sig}_{\text{prev,none}}$
  end if
else // we are `host_funder`
  $\text{sig}_{\text{next,none}} \leftarrow \textsc{sign}(\text{TX}_{\text{next,none}}, sk_{\text{loc},F})$
  $\text{sig}_{\text{next,right}} \leftarrow \textsc{sign}(\text{TX}_{\text{next,right}}, sk_{\text{loc},F})$
  call $\textsc{virt.circulateFundingSigs}(\text{sig}_{\text{next,none}}, \text{sig}_{\text{next,right}})$ of $\bar{P}$ and assign returned value to $\text{sig}_{\text{loc,none}}$
  ensure $\textsc{verify}(\text{TX}_{\text{loc,none}}, \text{sig}_{\text{loc,none}}, pk_{\text{next},F}) = \text{True}$ // $pk_{\text{next},F}$ found in `right_data`
  $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
  return ($\textsc{ok}$)
end if

Fig. 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 ← guest.revokePrevious()
    input (READ) to G_Ledger and assign ouput to Σ
    last_poll ← |Σ|
    call virt.circulateRevocations(revoc_for_next) of P̄ 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 ← guest.revokePrevious()
    input (READ) to G_Ledger and assign ouput to Σ
    last_poll ← |Σ|
    call virt.circulateRevocations(revoc_for_next) of P̄ and assign output to revoc_by_next
    ensure guest.processRemoteRevocation(revoc_by_next) returns (OK)
    output (HOSTS READY) to guest and expect reply (HOST ACK)
    output (VIRTUAL REVOCATION BACK)
end if
revoc_for_prev ← guest.revokePrevious()
output (HOSTS READY) to guest and expect reply (HOST ACK)
return revoc_for_prev // we are not host_fundee nor host_funder
```

Fig. 38.

Process VIRT – poll

On input (CHECK FOR LATERAL CLOSE) by $R \in \{$`guest`, `funder`, `fundee`$\}$:
  input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
  `prev_went_on_chain` $\leftarrow \text{TX}_{\text{prev,left}} \in \Sigma \vee \text{TX}_{\text{prev,none}} \in \Sigma$
  `next_went_on_chain` $\leftarrow \text{TX}_{\text{next,right}} \in \Sigma \vee \text{TX}_{\text{next,none}} \in \Sigma$
  `last_poll` $\leftarrow |\Sigma|$
  if `prev_went_on_chain` $\vee$ `next_went_on_chain` **then**
    ignore all messages except for (CHECK CHAIN FOR CLOSING) by R
    $State \leftarrow$ CLOSING
  end if
  if `prev_went_on_chain` $\wedge$ `next_went_on_chain` **then**
    VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,both}}$, $\text{sigs}_{\text{loc,both}}$)
  else if `prev_went_on_chain` **then**
    VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,left}}$, $\text{sigs}_{\text{loc,left}}$)
  else if `next_went_on_chain` **then**
    VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,right}}$, $\text{sigs}_{\text{loc,right}}$)
  end if

VIRT.SIGNANDSUBMIT(tx, sigs):
  add SIGN(tx, $sk_{\text{loc},F}$) to sigs
  input (SUBMIT, tx, sigs) to $\mathcal{G}_{\text{Ledger}}$

Fig. 39.

Process VIRT – close

On input (CLOSE) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$: // At most one of $\texttt{funder}$, $\texttt{fundee}$ is defined
    if $State$ = CLOSED **then** output (CLOSED) to R
    if $State$ = GUEST PUNISHED **then** output (GUEST PUNISHED) to R
    ensure $State$ = OPEN
    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 CHAIN FOR CLOSING) 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 $\texttt{funder}$ or $\texttt{fundee}$ is defined and $\text{TX}_{\text{none,loc}}$ is not valid in Σ **then** // we are an edge node and our counterparty has closed
        ignore all messages except for (CHECK CHAIN FOR CLOSING) by R
        $State \leftarrow$ CLOSING
        give up execution token // control goes to $\mathcal{E}$
    end if
    let $\texttt{tx}$ be the unique valid TX for Σ among ($\text{TX}_{\text{loc,none}}$, $\text{TX}_{\text{loc,left}}$, $\text{TX}_{\text{loc,right}}$, $\text{TX}_{\text{loc,both}}$) // if we are not an intermediary, only the first exists
    let $\texttt{sigs}$ be the corresponding set of signatures among ($\text{sigs}_{\text{loc,none}}$, $\text{sigs}_{\text{loc,left}}$, $\text{sigs}_{\text{loc,right}}$, $\text{sigs}_{\text{loc,both}}$)
    add SIGN($\texttt{tx}$, $sk_{A,F}$) and SIGN($\texttt{tx}$, $sk_{\text{loc,virt}}$) to $\texttt{sigs}$ // one of the two signatures may be empty, as some transactions don't need a signature by both keys. This is not a problem.
    ignore all messages except for (CHECK CHAIN FOR CLOSING) by R
    $State \leftarrow$ CLOSING
    send (SUBMIT, ($\texttt{tx}$, $\texttt{sigs}$)) to $\mathcal{G}_{\text{Ledger}}$

On (CHECK CHAIN FOR 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 $R = \texttt{guest}$ **then**
        $pk_1 \leftarrow pk_{\text{left,guest}}$; $pk_2 \leftarrow pk_{\text{right,guest}}$
    else // $R \in \{\texttt{funder}, \texttt{fundee}\}$
        $pk_1 \leftarrow pk_{\text{loc,virt}}$; $pk_2 \leftarrow pk_{\text{rem,virt}}$
    end if
    if Σ has an unspent output that can be spent exclusively by a 2-of-$\{pk_1, pk_2\}$ multisig **then** // if there is a timelock, it must have expired
        $State \leftarrow$ CLOSED
        output (CLOSED) to R
    end if

Fig. 40.

Process VIRT – punishment handling

```
On input (USED REVOCATION) by guest: // (USED REVOCATION) by funder/fundee is ignored
    State ← GUEST PUNISHED
    input (USED REVOCATION) to 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 ← GUEST PUNISHED
    output (ENABLER USED REVOCATION) to guest

On output (USED REVOCATION) by host_P:
    State ← 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
```

Fig. 41.

Lemma 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 $(c_i, pk_{P,\text{out}})$ outputs such that*

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

with overwhelming probability in the security parameter.

Proof. 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 (OPEN, c, `hops`, ...) 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 (FUNDED, ...) to P, as they were at the moment the latter message was sent. After initialised, F remains immutable. Observe that 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), $[-c_{\text{guest}}]$ 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 `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 (OPEN, c, ...) 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 (OPEN, ...) message by $\bar{P}$ (Fig. 10, l. 3), otherwise if $P =$ *Alice* $pk_{\bar{P},F}$ must have been contained in the (ACCEPT CHANNEL, ...) message by $\bar{P}$ (Fig. 10, l. 15).
- 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. 27, 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 with overwhelming probability (as a signature forgery may happen only with negligible probability), 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 (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' is $C + [-f]$) 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 $(c_i, pk_{P,\text{out}})$ transaction outputs such that* $\sum\limits_{i=1}^{h} c_i \geq \sum\limits_{s \in C} \sum\limits_{x \in s} x$. Furthermore, given that P moves to the OPEN state after the (FUND ME, $\dots$) message, it also sends (FUNDED, $\dots$) 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 $(c_i^R, pk_{R,\text{out}})$ transaction outputs such that* $\sum\limits_{i=1}^{k} c_i^R \geq \sum\limits_{s \in C_R} \sum\limits_{x \in s} x$.

– If P receives (VIRTUALISE, $\dots$) by an LN ITI R, subsequently transitions back to OPEN (therefore moving to history (F, C') where C' is $C + [0]$) 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 $(c_i, pk_{P,\text{out}})$ transaction outputs such that* $\sum\limits_{i=1}^{h} c_i \geq \sum\limits_{s \in C} \sum\limits_{i \in s} i$. Furthermore, given that P moves to the OPEN state after the (VIRTUALISE, $\dots$) message and in case it sends (FUNDED, $\dots$) 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* k $(c_i^R, pk_{R,\text{out}})$ *transaction outputs such that* $\sum\limits_{i=1}^{k} c_i^R \geq \sum\limits_{s \in C_R} \sum\limits_{x \in s} x$.

- 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 $F = []$) before any further change to its history, then *eventually* P*'s* $\mathcal{G}_{\text{Ledger}}$ *state will contain* h $(c_i, pk_{P,\text{out}})$ *transaction outputs such that* $\sum\limits_{i=1}^{h} c_i \geq \sum\limits_{s \in C} \sum\limits_{i \in s} i$.
- 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 $(c_i, pk_{P,\text{out}})$ *transaction outputs such that* $\sum\limits_{i=1}^{h} c_i \geq \sum\limits_{s \in C} \sum\limits_{i \in s} i$.

When P is in the OPEN state and receives (FUND ME, $f, \dots$), it can only move again to the OPEN state after doing the following state transitions: OPEN $\rightarrow$ VIRTUALISING $\rightarrow$ WAITING FOR REVOCATION $\rightarrow$ WAITING FOR INBOUND REVOCATION $\rightarrow$ WAITING FOR HOSTS READY $\rightarrow$ OPEN. During this sequence of events, a new 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 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 that 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). When P receives (CLOSE) by $\mathcal{E}$, it inputs (CLOSE) to $\texttt{host}_P$ (Fig. 27, l. 4). If the $\texttt{host}$ of $\texttt{host}_P$ is $\mathcal{G}_{\text{Ledger}}$, then $\texttt{host}_P$ either is able to publish its $\text{TX}_{\text{loc,none}}$ (since it has necessarily received a valid $\text{sig}_{\text{loc,none}}$ (Fig. 37, l. 33) by its counterparty before it moved to the OPEN state for the first time), or the output needed to spend $\text{TX}_{\text{loc,none}}$ has already been spent. The only other transactions that can spend it are $\text{TX}_{2,\text{none}}$ and $\text{TX}_{2,\text{right}}$, since these are the only transactions that spend the aforementioned output and that $\texttt{host}_P$ has signed (Fig. 37, ll. 30 and 31). The output can be also spent by old, revoked commitment transactions, but in that case $\texttt{host}_P$ would not have output (CLOSED). If $\text{TX}_{\text{loc,none}}$ ends up in $\mathcal{G}_{\text{Ledger}}$, its $\texttt{out}$ output (Fig. 31, l. 9) carries $c_P + c_{\bar{P}} - f$ coins and can be spent either by a timelocked $\texttt{3} \wedge 2/\{pk'_{P,F}, pk'_{\bar{P},F}\}$ spending method or by a non-timelocked $\texttt{1} \wedge 4/\{pk'_{P,F}, pk'_{\bar{P},F}, pk_{R,F}, pk_{\bar{R},F}\}$ spending method. The only transactions with an input having the latter spending method that $\texttt{host}_P$ has signed with $sk'_{P,F}$ are $\text{TX}_{2,\text{left}}$ and $\text{TX}_{2,\text{both}}$ (Fig. 35) and they both have a $(c_P + c_{\bar{P}} - f,\ \texttt{3} \wedge 2/\{pk'_{P,F}, pk'_{\bar{P},F}\})$ output. Similarly, both $\text{TX}_{2,\text{none}}$ and $\text{TX}_{2,\text{right}}$ have an output with $c_P + c_{\bar{P}} - f$ coins that can be spent either by a timelocked $\texttt{3} \wedge 2/\{pk'_{P,F}, pk'_{\bar{P},F}\}$ spending method or by a non-timelocked

$2 \wedge 4/\{pk'_{P,F}, pk'_{\bar{P},F}, pk_{R,F}, pk_{\bar{R},F}\}$ spending method. $\texttt{host}_P$ has never signed any transaction that can use the latter spending method, therefore the timelock will eventually expire. To sum up, in all cases a $(c_P + c_{\bar{P}} - f,\ 3 \wedge 2/\{pk'_{P,F}, pk'_{\bar{P},F}\})$ output will end up in $\mathcal{G}_{\text{Ledger}}$. This output can 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). By the induction hypothesis, before the funding procedure started P could close the channel for an $(c_P,\ pk_{P,\text{out}})$ output. We have now proven that if P completes the funding of a new channel and it has $F = []$, then it can close its channel for a $(c_P - f,\ pk_{P,\text{out}})$ output. Regarding the case that $F \neq []$, this means that before the funding procedure started $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$. By the inductive hypothesis, P could then close the channel for an $(c_P,\ pk_{P,\text{out}})$ output. 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 as before 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 intrerface than before, therefore it will successfully close by the induction hypothesis. Thereafter the process is identical to the one when the old $\texttt{host}_P = \mathcal{G}_{\text{Ledger}}$. We have therefore proven the first claim of the first bullet. □

Lemma 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, $c, \dots$) by $\mathcal{E}$ prior to transitioning to the OPEN State,*
- *P [has received (FUND ME, $f_i, \dots$) 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:

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

Proof. 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, ...) 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 29). 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. 22). 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 3 (No halt). *In an ideal execution with $\mathcal{F}_{\text{Chan}}$ and $\mathcal{S}$, the functionality halts with negligible probability in the security parameter (i.e. l. 9 of Fig. 5 is executed negligibly often).*

Proof. We prove the Lemma in two steps. We first show that if the conditions of Lemma 2 hold, then the conditions of Lemma 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 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. 41, 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 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, $\mathcal{G}_{\text{Ledger}}$, $\ldots$) by $\mathcal{E}$ and (BASE OPEN) by $\mathcal{S}$ or (OPEN, c, `hops` ($\neq \mathcal{G}_{\text{Ledger}}$), $\ldots$) by $\mathcal{E}$, (FUNDED, HOST, $\ldots$) by `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 2 holds and $P = Alice$, then the second condition of Lemma 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 2 holds only if the second condition of Lemma 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 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, $\ldots$) 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, $\ldots$) 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 1 holds with the same n as that of Lemma 2.

Regarding the fourth Lemma 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 1 holds with the same m as that of Lemma 2. As far as the fifth condition of Lemma 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 2 holds with some l, then Lemma 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. 6), we deduce that if the conditions of Lemma 2 hold, then the functionality halts only with negligible probability.

In the second proof step, we show that if the conditions of Lemma 2 do not hold, then the check of Fig. 5, l. 6 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. 3). 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. 3 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 2 are always satisfied. We conclude that if the conditions to Lemma 2 do not hold, then the check of Fig. 5, l. 6 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 1 (Recursive Virtual Payment Channel Security). *The protocol LN realises $\mathcal{F}_{\text{Chan}}$ with simulator $\mathcal{S}$ given a global functionality $\mathcal{G}_{\text{Ledger}}$ and assuming the security of the underlying digital signature. Specifically,*

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

Proof. 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 LN parties exactly as they would execute in the real world execution, $\textsc{exec}^{\mathcal{G}_{\text{Ledger}}}_{\text{LN},\mathcal{A},\mathcal{E}}$

in case $\mathcal{F}_{\text{Chan}}$ does not halt. Indeed, $\mathcal{F}_{\text{Chan}}$ only halts with negligible probability according to Lemma 3, therefore the two executions are computationally indistinguishable. $\square$

References