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 $P \in \{Alice, Bob\}$

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

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

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

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

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

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

17: On (OPEN, $x$, $\texttt{hops} \neq \mathcal{G}_{\text{Ledger}}$, $\dots$) to $Alice$ by $\mathcal{E}$ when $State_A =$ INIT:
18:    store $x$
19:    $\texttt{enabler}_A \leftarrow \texttt{hops}[0].\texttt{left}$
20:    $State_A \leftarrow$ PENDING VIRTUAL OPEN

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

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

28: On (VIRTUAL OPEN) by $\mathcal{A}$ when $State_P =$ TENTATIVE VIRTUAL OPEN:
29:    if $P = Alice$ then $\texttt{balance}_P \leftarrow x$
30:    $State_P \leftarrow$ 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 $\mathcal{F}_{\text{Chan}}$ – fundings state machine for $P \in \{Alice, Bob\}$

```
1: On input (FUND ME, $x$, ...) by ITI $R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$ 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 $State_{\bar{P}} \in \{$IGNORED, SYNC HELP FUND$\}$ then
22:         balance$_P \leftarrow$ balance$_P - x$
23:         host$_P \leftarrow$ 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 $State_{\bar{P}} \in \{$IGNORED, SYNC FUND$\}$ then
29:         host$_P \leftarrow$ 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 $P \in \{Alice, Bob\}$

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

3: On (CLOSE, $P$) by $\mathcal{A}$ when $State \notin$ {UNINIT, INIT, PENDING VIRTUAL OPEN,
   TENTATIVE VIRTUAL OPEN, TENTATIVE BASE OPEN, IGNORED}:
4:     input (READ) to $\mathcal{G}_{\text{Ledger}}$ as $P$ and assign ouput to $\Sigma$
5:     $\texttt{coins}_P \leftarrow$ sum of coins exclusively spendable by $pk_P$ in $\Sigma$
6:     if $\texttt{coins}_P \geq \texttt{balance}_P$ then
7:         $State_P \leftarrow$ CLOSED
8:     else // balance security is broken
9:         halt
10:    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.

```
1: On (CORRUPT) by $\mathcal{A}$, addresed to $P$:
2:     // 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.
3:     send (INFO, BECAME CORRUPTED OR NEGLIGENT, $P$) to $\mathcal{F}_{\text{Chan}}$

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

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

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

10: 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):
11:     if hops $= \mathcal{G}_{\text{Ledger}}$ then
12:         send (INFO, BASE OPEN) to $\mathcal{F}_{\text{Chan}}$
13:     else
14:         send (INFO, VIRTUAL OPEN) to $\mathcal{F}_{\text{Chan}}$
15:     end if

16: 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
17:     send (INFO, PAY) to $\mathcal{F}_{\text{Chan}}$

18: 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
19:     send (INFO, FUND) to $\mathcal{F}_{\text{Chan}}$

20: 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):
21:     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 $\texttt{last\_poll} \neq \bot$ then // channel has opened
        input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to $\Sigma$
        if $\texttt{last\_poll} + t < |\Sigma|$ then
            $\texttt{negligent} \leftarrow$ True
        end if
    end if

On (INIT, $pk_{P,\text{out}}$):
    ensure $State = \bot$
    $State \leftarrow$ INIT
    store $pk_{P,\text{out}}$
    $(c_A, c_B, \texttt{locked}_A, \texttt{locked}_B) \leftarrow (0, 0, 0, 0)$
    $(\texttt{paid\_out}, \texttt{paid\_in}) \leftarrow (\emptyset, \emptyset)$
    $\texttt{negligent} \leftarrow$ False
    $\texttt{last\_poll} \leftarrow \bot$
    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 $\Sigma$
    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 $\Sigma$
    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

```
1: revokePrevious():
2:     ensure $State \in$ waiting for (outbound) revocation
3:     $R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$P$, output: ($C_{P,i}$.outputs.$P$.value,
   $pk_{\bar{P},\text{out}}$)}
4:     $\text{sig}_{A,R,i} \leftarrow$ sign($R_{\bar{P},i}$, $sk_{P,R}$)
5:     if $State =$ waiting for revocation then
6:         $State \leftarrow$ waiting for inbound revocation
7:     else // $State =$ waiting for outbound revocation
8:         $i \leftarrow i + 1$
9:         $State \leftarrow$ waiting for hosts ready
10:    end if
11:    $\texttt{host}_P \leftarrow \texttt{host}'_P$ // forget old host, use new host instead
12:    $\texttt{layer} \leftarrow \texttt{layer} + 1$
13:    return $\text{sig}_{P,R,i}$

14: processRemoteRevocation($\text{sig}_{\bar{P},R,i}$):
15:    ensure $State =$ waiting for (inbound) revocation
16:    $R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.$P$, output: ($C_{\bar{P},i}$.outputs.$\bar{P}$.value,
   $pk_{P,\text{out}}$)}
17:    ensure verify($R_{P,i}$, $\text{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
18:    if $State =$ waiting for revocation then
19:        $State \leftarrow$ waiting for outbound revocation
20:    else // $State =$ waiting for inbound revocation
21:        $i \leftarrow i + 1$
22:        $State \leftarrow$ waiting for hosts ready
23:    end if
24:    return (ok)

25: negligent():
26:    $\texttt{negligent} \leftarrow$ True
27:    return (ok)
```

Fig. 9.

Process LN.EXCHANGEOPENKEYS()

```
1: $(sk_{A,F}, pk_{A,F}) \leftarrow \textsc{keyGen}(); (sk_{A,R}, pk_{A,R}) \leftarrow \textsc{keyGen}()$
2: $State \leftarrow$ WAITING FOR OPENING KEYS
3: send (OPEN, $c$, hops, $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$) to fundee
4: // colored code is run by honest fundee. Validation is implicit
5: ensure we run the code of $Bob$
6: ensure $State =$ INIT
7: store $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$
8: $(sk_{B,F}, pk_{B,F}) \leftarrow \textsc{keyGen}(); (sk_{B,R}, pk_{B,R}) \leftarrow \textsc{keyGen}()$
9: if hops $= \mathcal{G}_{\text{Ledger}}$ then // opening base channel
10:     layer $\leftarrow 0$
11:     $State \leftarrow$ WAITING FOR COMM SIG
12: else // opening virtual channel
13:     $State \leftarrow$ WAITING FOR CHECK KEYS
14: end if
15: reply (ACCEPT CHANNEL, $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$)
16: ensure $State =$ WAITING FOR OPENING KEYS
17: store $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$
18: $State \leftarrow$ OPENING KEYS OK
```

Fig. 10.

Process LN.PREPAREBASE()

```
1: if hops $= \mathcal{G}_{\text{Ledger}}$ then // opening base channel
2:     $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}
3:     host$_P \leftarrow \mathcal{G}_{\text{Ledger}}$
4:     layer $\leftarrow 0$
5: else // opening virtual channel
6:     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
7:     layer $\leftarrow$ funder_layer
8: end if
```

Fig. 11.

```
Process LN.EXCHANGEOPENSIGS()
1:  // $s = (2 + \lceil \texttt{maxTime}_{\texttt{window}} + \frac{\texttt{Delay}}{2} / \texttt{minTime}_{\texttt{window}} \rceil)\texttt{windowSize}$, where
    $\texttt{maxTime}_{\texttt{window}}$, Delay, $\texttt{minTime}_{\texttt{window}}$ and windowSize are defined in
    Proposition ?? TODO: recheck and include proposition
2:  $C_{A,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, (pk_{A,\text{out}} + (t+s)) \vee$
    $2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}
3:  $C_{B,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, pk_{A,\text{out}})$, $(0,$
    $(pk_{B,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}
4:  $\text{sig}_{A,C,0} \leftarrow \text{SIGN}(C_{B,0}, sk_{A,F})$
5:  $State \leftarrow$ WAITING FOR COMM SIG
6:  send (FUNDING CREATED, $(c, pk_{A,\text{chain}})$, $\text{sig}_{A,C,0}$) to fundee
7:  ensure $State =$ WAITING FOR COMM SIG // if opening virtual channel, we have
    received (FUNDED, host_fundee) by hops[-1].right (Fig 14, l. 10)
8:  if hops $= \mathcal{G}_{\text{Ledger}}$ then // opening base channel
9:      $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}
10: end if
11: $C_{B,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, pk_{A,\text{out}})$, $(0,$
    $(pk_{B,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}
12: ensure $\text{VERIFY}(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F}) =$ True
13: $C_{A,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, (pk_{A,\text{out}} + (t+s)) \vee$
    $2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}
14: $\text{sig}_{B,C,0} \leftarrow \text{SIGN}(C_{A,0}, sk_{B,F})$
15: if hops $= \mathcal{G}_{\text{Ledger}}$ then // opening base channel
16:     $State \leftarrow$ WAITING TO CHECK FUNDING
17: else // opening virtual channel
18:     $c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
19:     $State \leftarrow$ OPEN
20: end if
21: reply (FUNDING SIGNED, $\text{sig}_{B,C,0}$)
22: ensure $State =$ WAITING FOR COMM SIG
23: ensure $\text{VERIFY}(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F}) =$ True
```

Fig. 12.

```
Process LN.COMMITBASE()
1: $\text{sig}_F \leftarrow \text{SIGN}(F, sk_{A,\text{chain}})$
2: send (OPEN, $c, pk_{A,\text{out}}, pk_{B,\text{out}}, F, \text{sig}_F, Alice, Bob$) to $\mathcal{A}$
3: while $F \notin \Sigma$ do
4:     wait for input (CHECK FUNDING) // ignore all other messages
5:     input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to $\Sigma$
6: end while
```

Fig. 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 $\Sigma$
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) by hops[-1].right:
9:      ensure $State$ = WAITING FOR FUNDED
10:     store host_P // we will talk directly to host_P
11:     layer ← 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)
```

Fig. 14.

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

```
1:  // fundee is Bob
2:  ensure we run the code of Alice // activated party is the funder
3:  if hops = $\mathcal{G}_{\text{Ledger}}$ then // opening base channel
4:      ensure $State$ = TOPPED UP
5:      ensure $c = c_{A,\text{chain}}$
6:  else // opening virtual channel
7:      ensure len(hops) $\geq 2$ // cannot open a virtual over 1 channel
8:  end if
9:  LN.EXCHANGEOPENKEYS()
10: LN.PREPAREBASE()
11: LN.EXCHANGEOPENSIGS()
12: if hops = $\mathcal{G}_{\text{Ledger}}$ then
13:     LN.COMMITBASE()
14: end if
15: input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to $\Sigma$
16: last_poll $\leftarrow |\Sigma|$
17: $c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
18: $State \leftarrow$ OPEN
19: output (OPEN OK, $c$, fundee, hops)
```

Fig. 15.

Process LN.UPDATEFORVIRTUAL()

```
1:  $C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with $pk'_{P,F}$ and $pk'_{\bar{P},F}$ instead of $pk_{P,F}$ and $pk_{\bar{P},F}$ respectively
2:  $\text{sig}_{P,C,i+1} \leftarrow$ SIGN($C_{\bar{P},i+1}$) // kept by $\bar{P}$
3:  send (UPDATE FORWARD, $\text{sig}_{P,C,i+1}$) to $\bar{P}$
4:  // $P$ refers to payer and $\bar{P}$ to payee both in local and remote code
5:  $C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with $pk'_{P,F}$ and $pk'_{\bar{P},F}$ instead of $pk_{P,F}$ and $pk_{\bar{P},F}$ respectively
6:  ensure VERIFY($C_{\bar{P},i+1}$, $\text{sig}_{P,C,i+1}$, $pk'_{P,F}$) = True
7:  $C_{P,i+1} \leftarrow C_{P,i}$ with $pk'_{\bar{P},F}$ and $pk'_{P,F}$ instead of $pk_{\bar{P},F}$ and $pk_{P,F}$ respectively
8:  $\text{sig}_{\bar{P},C,i+1} \leftarrow$ SIGN($C_{P,i+1}, sk'_{\bar{P},F}$) // kept by $P$
9:  reply (UPDATE BACK, $\text{sig}_{\bar{P},C,i+1}$)
10: $C_{P,i+1} \leftarrow C_{P,i}$ with $pk'_{\bar{P},F}$ and $pk'_{P,F}$ instead of $pk_{\bar{P},F}$ and $pk_{P,F}$ respectively
11: 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

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

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

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

Fig. 17.

Process LN – virtualise hops

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

17: On input (VIRTUALISING, host_sibling, $(sk'_{P,F}$, $pk'_{P,F})$, $pk_{\text{sib},\bar{P},F}$, hops,
    fundee, $c_{\text{guest}}$, $c_{\text{sib,rem}}$, $_{\text{sib}}$) by sibling:
18:    ensure $State$ = OPEN
19:    ensure $c_P - \texttt{locked}_P \geq c$
20:    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
21:    $State \leftarrow$ VIRTUALISING
22:    $\texttt{locked}_P \leftarrow \texttt{locked}_P + c$
23:    define new VIRT ITI $\texttt{host}'_P$
24:    send (VIRTUALISING, $\texttt{host}'_P$, $pk'_{P,F}$, hops, fundee, $c_{\text{guest}}$) to hops[0].right
       and expect reply (VIRTUALISING ACK, $\texttt{host}'_{\bar{P}}$, $pk'_{\bar{P},F}$)
25:    ensure $pk'_{\bar{P},F}$ is different from $pk_{\bar{P},F}$ and all older $\bar{P}$'s funding public keys
26:    LN.UPDATEFORVIRTUAL()
27:    input (INIT, $\texttt{host}_P$, $\texttt{host}'_{\bar{P}}$, host_sibling, $(sk'_{P,F}$, $pk'_{P,F})$, $pk'_{\bar{P},F}$, $pk_{\text{sib},\bar{P},F}$,
       $(sk_{P,F}$, $pk_{P,F})$, $pk_{\bar{P},F}$, $c_{\text{guest}}$) to $\texttt{host}'_P$ and expect reply (HOST INIT OK)
28:    $State \leftarrow$ WAITING FOR REVOCATION
29:    output (VIRTUALISING ACK, $\texttt{host}'_P$, $pk'_{\bar{P},F}$) to sibling
```

Fig. 18.

```
Process LN.SIGNATURESROUNDTRIP()
1: $C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with $x$ coins moved from $P$'s to $\bar{P}$'s output
2: $\text{sig}_{P,C,i+1} \leftarrow \text{SIGN}(C_{\bar{P},i+1}, sk_{P,F})$ // kept by $\bar{P}$
3: $State \leftarrow$ WAITING FOR COMMITMENT SIGNED
4: send (PAY, $x$, $\text{sig}_{P,C,i+1}$) to $\bar{P}$
5: // $P$ refers to payer and $\bar{P}$ to payee both in local and remote code
6: ensure $State =$ WAITING TO GET PAID $\wedge x = y$
7: if $\texttt{host}_{\bar{P}} \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$ has a host_sibling then // we are intermediary channel
8:     ensure $c_{\text{sib,rem}} \geq c_P - x \wedge c_{\bar{P}} + x \geq c_{\text{sib}}$ // avoid value loss by griefing attack
9: end if
10: $C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with $x$ coins moved from $P$'s to $\bar{P}$'s output
11: ensure $\text{VERIFY}(C_{\bar{P},i+1}, \text{sig}_{P,C,i+1}, pk_{P,F}) =$ True
12: $C_{P,i+1} \leftarrow C_{P,i}$ with $x$ coins moved from $P$'s to $\bar{P}$'s output
13: $\text{sig}_{\bar{P},C,i+1} \leftarrow \text{SIGN}(C_{P,i+1}, sk_{\bar{P},F})$ // kept by $P$
14: $R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.$P$, output: $(c_{\bar{P}}, pk_{P,\text{out}})$}
15: $\text{sig}_{\bar{P},R,i} \leftarrow \text{SIGN}(R_{P,i}, sk_{\bar{P},R})$
16: $State \leftarrow$ WAITING FOR PAY REVOCATION
17: reply (COMMITMENT SIGNED, $\text{sig}_{\bar{P},C,i+1}$, $\text{sig}_{\bar{P},R,i}$)
18: ensure $State =$ WAITING FOR COMMITMENT SIGNED
19: $C_{P,i+1} \leftarrow C_{P,i}$ with $x$ coins moved from $P$'s to $\bar{P}$'s output
```

Fig. 19.

Process LN.REVOCATIONSTRIP()

```
1: ensure VERIFY($C_{P,i+1}$, $\text{sig}_{\bar{P},C,i+1}$, $pk_{\bar{P},F}$) = True
2: $R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.$P$, output: ($c_{\bar{P}}$, $pk_{P,\text{out}}$)}
3: ensure VERIFY($R_{P,i}$, $\text{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
4: $R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: ($c_P$, $pk_{\bar{P},\text{out}}$)}
5: $\text{sig}_{P,R,i} \leftarrow$ SIGN($R_{\bar{P},i}$, $sk_{P,R}$)
6: add $x$ to paid_out
7: $c_P \leftarrow c_P - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x$; $i \leftarrow i + 1$
8: $State \leftarrow$ OPEN
9: if host$_P \neq \mathcal{G}_{\text{Ledger}} \wedge$ we have a host_sibling then // we are intermediary channel
10:     input (NEW BALANCE, $c_P$, $c_{\bar{P}}$) to host$_P$
11:     relay message as input to sibling // run by VIRT
12:     relay message as output to guest // run by VIRT
13:     store new sibling balance and reply (NEW BALANCE OK)
14:     output (NEW BALANCE OK) to sibling // run by VIRT
15:     output (NEW BALANCE OK) to guest // run by VIRT
16: end if
17: send (REVOKE AND ACK, $\text{sig}_{P,R,i}$) to $\bar{P}$
18: ensure $State$ = WAITING FOR PAY REVOCATION
19: $R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: ($c_P$, $pk_{\bar{P},\text{out}}$)}
20: ensure VERIFY($R_{\bar{P},i}$, $\text{sig}_{P,R,i}$, $pk_{P,R}$) = True
21: add $x$ to paid_in
22: $c_P \leftarrow c_P - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x$; $i \leftarrow i + 1$
23: $State \leftarrow$ OPEN
24: if host$_P \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$ has a host_sibling then // we are intermediary channel
25:     input (NEW BALANCE, $c_{\bar{P}}$, $c_P$) to host$_{\bar{P}}$
26:     relay message as input to sibling // run by VIRT
27:     relay message as output to guest // run by VIRT
28:     store new sibling balance and reply (NEW BALANCE OK)
29:     output (NEW BALANCE OK) to sibling // run by VIRT
30:     output (NEW BALANCE OK) to guest // run by VIRT
31: end if
```

Fig. 20.

```
Process LN – On (PAY, $x$):
1: ensure $State = $ OPEN $\wedge c_P \geq x$
2: if host$_P \neq \mathcal{G}_{\text{Ledger}} \wedge P$ has a host_sibling then // we are intermediary channel
3:     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
4: end if
5: LN.SIGNATURESROUNDTRIP()
6: LN.REVOCATIONSTRIP()
7: // No output is given to the caller, this is intentional
```

Fig. 21.

```
Process LN – On (GET PAID, $y$):
1: ensure $State = $ OPEN $\wedge c_{\bar{P}} \geq x$
2: store $y$
3: $State \leftarrow$ WAITING TO GET PAID
```

Fig. 22.

```
Process LN – On (CHECK FOR LATERAL CLOSE):
1: if host$_P \neq \mathcal{G}_{\text{Ledger}}$ then
2:     input (CHECK FOR LATERAL CLOSE) to host$_P$
3: end if
```

Fig. 23.

Process LN – On (CHECK CHAIN FOR CLOSED):

```
1: ensure $State \notin \{\bot, \textsc{init}, \textsc{topped up}\}$ // channel open
2: // even virtual channels check $\mathcal{G}_{\text{Ledger}}$ directly. This is intentional
3: input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to $\Sigma$
4: last_poll $\leftarrow |\Sigma|$
5: if $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$ then // counterparty has closed maliciously
6:     $State \leftarrow$ CLOSING
7:     LN.SUBMITANDCHECKREVOCATION$(j)$
8:     $State \leftarrow$ CLOSED
9:     output (CLOSED)
10: else if $C_{P,j} \in \Sigma \land C_{\bar{P},j} \in \Sigma$ then
11:     $State \leftarrow$ CLOSED
12:     output (CLOSED)
13: end if
```

Fig. 24.

Process LN.SUBMITANDCHECKREVOCATION(j)

```
1: $\text{sig}_{P,R,j} \leftarrow \textsc{sign}(R_{P,j}, sk_{P,R})$
2: input (SUBMIT, $(R_{P,j}, \text{sig}_{P,R,j}, \text{sig}_{\bar{P},R,j})$) to $\mathcal{G}_{\text{Ledger}}$
3: while $\nexists R_{P,j} \in \Sigma$ do
4:     wait for input (CHECK REVOCATION) // ignore other messages
5:     input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to $\Sigma$
6: end while
7: $c_P \leftarrow c_P + c_{\bar{P}}$
8: if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ then
9:     input (USED REVOCATION) to $\texttt{host}_P$
10: end if
```

Fig. 25.

```
Process LN – On (CLOSE):
1: ensure $State \notin \{\bot, \text{INIT}, \text{TOPPED UP}, \text{CLOSED}, \text{BASE PUNISHED}\}$ // channel open
2: if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ then // we have a virtual channel
3:     $State \leftarrow$ HOST CLOSING
4:     input (CLOSE) to $\texttt{host}_P$ and keep relaying inputs (CHECK CHAIN FOR
   CLOSING) to $\texttt{host}_P$ until receiving output (CLOSED) by $\texttt{host}_P$
5:     $\texttt{host}_P \leftarrow \mathcal{G}_{\text{Ledger}}$
6: end if
7: $State \leftarrow$ CLOSING
8: input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to $\Sigma$
9: if $C_{\bar{P},i} \in \Sigma$ then // counterparty has closed honestly
10:     no-op // do nothing
11: else if $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$ then // counterparty has closed maliciously
12:     LN.SUBMITANDCHECKREVOCATION($j$)
13: else // counterparty is idle
14:     while $\nexists$ unspent output $\in \Sigma$ that $C_{P,i}$ can spend do // possibly due to an
   active timelock
15:         wait for input (CHECK VIRTUAL) // ignore other messages
16:         input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to $\Sigma$
17:     end while
18:     $\text{sig}'_{P,C,i} \leftarrow \text{SIGN}(C_{P,i}, sk_{P,F})$
19:     input (SUBMIT, $(C_{P,i}, \text{sig}_{P,C,i}, \text{sig}'_{P,C,i})$) to $\mathcal{G}_{\text{Ledger}}$
20: end if
```

Fig. 26.

```
Process LN – On output (ENABLER USED REVOCATION) by $\texttt{host}_P$:
1: $State \leftarrow$ BASE PUNISHED
```

Fig. 27.

Process VIRT

```
1:  On every activation, before handling the message:
2:      if last_poll $\neq \bot$ then // virtual layer is ready
3:          input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to $\Sigma$
4:          if last_poll $+ t < |\Sigma|$ then
5:              for $P \in \{$guest, funder, fundee$\}$ do // at most 1 of funder, fundee is defined
6:                  ensure $P$.NEGLIGENT() returns (OK)
7:              end for
8:          end if
9:      end if

10: // guest is trusted to give sane inputs, therefore a state machine and input verification is redundant
11: On input (INIT, host$_P$, $\bar{P}$, sibling, fundee, ($sk_{\text{loc,virt}}$, $pk_{\text{loc,virt}}$), $pk_{\text{rem,virt}}$, $pk_{\text{sib,rem,virt}}$, ($sk_{\text{loc},F}$, $pk_{\text{loc},F}$), $pk_{\text{rem},F}$, $c_{\text{guest}}$) by guest:
12:     store message contents and guest // sibling, $pk_{\text{sib},\bar{P},F}$ are missing for edge nodes, fundee is present only in last node
13:     last_poll $\leftarrow \bot$
14:     output (HOST INIT OK) to guest

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

Fig. 28.

Process VIRT.CIRCULATEKEYSANDCOINS(`left_data`):

```
1: if left_data is given as argument then // we are not host_funder
2:     if we have a sibling then // we are not host_fundee
3:         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
4:         store input as left_data
5:         parse left_data as far_left_data, ($sk_{\mathrm{loc,virt}}$, $pk_{\mathrm{loc,virt}}$), ($sk_{\mathrm{sib},F}$, $pk_{\mathrm{sib},F}$), $pk_{\mathrm{sib,rem},F}$, $c_{\mathrm{sib}}$, $c_{\mathrm{sib,rem}}$ // remove parentheses as necessary
6:         call VIRT.CIRCULATEKEYSANDCOINS(left_data) of $\bar{P}$ and assign returned value to right_data
7:         parse right_data as far_right_data, $pk_{\mathrm{rem,virt}}$
8:         output (KEYS AND COINS BACK, right_data, ($sk_{\mathrm{loc},F}$, $pk_{\mathrm{loc},F}$), $pk_{\mathrm{rem},F}$, $c_P$, $c_{\bar{P}}$)
9:         store output as right_data
10:        parse right_data as far_right_data, ($sk_{\mathrm{sib},F}$, $pk_{\mathrm{sib},F}$), $pk_{\mathrm{sib,rem},F}$, $c_{\mathrm{sib}}$, $c_{\mathrm{sib,rem}}$
11:        return (right_data, $pk_{\mathrm{loc,virt}}$)
12:    else // we are host_fundee
13:        extract ($pk_{\mathrm{left,guest}}$, $pk_{\mathrm{right,guest}}$) from left_data
14:        output (CHECK KEYS, ($pk_{\mathrm{left,guest}}$, $pk_{\mathrm{right,guest}}$)) to fundee and expect reply (KEYS OK)
15:        return $pk_{\mathrm{loc,virt}}$
16:    end if
17: else // we are host_funder
18:    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
19:    return (OK)
20: end if
```

Fig. 29.

```
Process VIRT
1: getMidTXs($c_{\text{guest}}$, $c_{\text{loc}}$, $c_{\text{rem}}$, $c_{\text{sib}}$, $c_{\text{sibRem}}$, $pk_{\text{left,fund}}$, $pk_{\text{loc,fund}}$, $pk_{\text{sib,fund}}$,
   $pk_{\text{right,fund}}$, $pk_{\text{left,virt}}$, $pk_{\text{loc,virt}}$, $pk_{\text{sib,virt}}$, $pk_{\text{right,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$,
   $pk_{\text{loc,out}}$, $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$):
2:     ensure $c_{\text{sibRem}} \geq c_{\text{guest}} \wedge c_{\text{loc}} \geq c_{\text{guest}}$
3:     $c_{\text{left}} \leftarrow c_{\text{sib}} + c_{\text{sibRem}}$; $c_{\text{right}} \leftarrow c_{\text{loc}} + c_{\text{rem}}$
4:     left_fund $\leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{loc,fund}}\}$
5:     right_fund $\leftarrow 2/\{pk_{\text{sib,fund}}, pk_{\text{right,fund}}\}$
6:     left_virt $\leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}\}$
7:     left_virt_checked $\leftarrow 4/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
8:     right_virt $\leftarrow 2/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}\}$
9:     right_virt_checked $\leftarrow 4/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
10:    left_out_checked $\leftarrow$ (2 $\wedge$ left_virt_checked) $\vee$ (3 $\wedge$ left_virt + $(t + s)$)
11:    right_out $\leftarrow$ (1 $\wedge$ right_virt) $\vee$ (3 $\wedge$ right_virt + $(t + s)$)
12:
   right_out_checked $\leftarrow$ (1 $\wedge$ right_virt_checked) $\vee$ (3 $\wedge$ right_virt + $(t + s)$)
13:    guest_all $\leftarrow$ 5 $\wedge$ $n/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1 \dots n}\}\}$
14:    guest_out $\leftarrow$ 4 $\wedge$ $2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
15:    guest $\leftarrow$ (guest_out + $(t + s)$) $\vee$ guest_all
16:    $\text{TX}_{\text{none}} \leftarrow$ TX {inputs: (($c_{\text{left}}$, left_fund), ($c_{\text{right}}$, right_fund)), outputs:
   (($c_{\text{left}} - c_{\text{guest}}$, left_out_checked), ($c_{\text{right}} - c_{\text{guest}}$, right_out_checked),
   ($c_{\text{guest}}$, $pk_{\text{loc,out}}$), ($c_{\text{guest}}$, guest))}
17:    $\text{TX}_{\text{left}} \leftarrow$ TX {inputs: (($c_{\text{left}} - c_{\text{guest}}$, 1 $\wedge$ left_virt_checked), ($c_{\text{right}}$,
   right_fund)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, 3 $\wedge$ left_virt), ($c_{\text{right}} - c_{\text{guest}}$,
   right_out_checked), ($c_{\text{guest}}$, $pk_{\text{loc,out}}$))}
18:    $\text{TX}_{\text{right}} \leftarrow$ TX {inputs: (($c_{\text{left}}$, left_fund), ($c_{\text{right}} - c_{\text{guest}}$, 2 $\wedge$
   right_virt_checked), ($c_{\text{guest}}$, guest_all)), outputs: (($c_{\text{left}} - c_{\text{guest}}$,
   left_out_checked), ($c_{\text{right}} - c_{\text{guest}}$, 3 $\wedge$ right_virt), ($c_{\text{guest}}$, $pk_{\text{loc,out}}$), ($c_{\text{guest}}$,
   guest))}
19:    $\text{TX}_{\text{both}} \leftarrow$ TX {inputs: (($c_{\text{left}} - c_{\text{guest}}$, 1 $\wedge$ left_virt_checked),
   ($c_{\text{right}} - c_{\text{guest}}$, 2 $\wedge$ right_virt_checked), ($c_{\text{guest}}$, guest_all)), outputs:
   (($c_{\text{left}} - c_{\text{guest}}$, 3 $\wedge$ left_virt), ($c_{\text{right}} - c_{\text{guest}}$, 3 $\wedge$ right_virt),
   ($c_{\text{guest}}$, $pk_{\text{loc,out}}$)}
20:    **return** ($\text{TX}_{\text{none}}$, $\text{TX}_{\text{left}}$, $\text{TX}_{\text{right}}$, $\text{TX}_{\text{both}}$)
```

Fig. 30.

Process VIRT

```
1: // 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.
2: GETEDGETXS($c_{\text{guest}}$, $c_{\text{left}}$, $c_{\text{right}}$, $pk_{\text{left,fund}}$, $pk_{\text{right,fund}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$,
   $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$, is_funder):
3:     ensure $c_{\text{left}} \geq c_{\text{guest}}$
4:     $c_{\text{tot}} \leftarrow c_{\text{left}} + c_{\text{right}}$
5:     fund $\leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{right,fund}}\}$
6:     virt $\leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}\}$
7:     virt_checked $\leftarrow 4/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
8:     if is_funder = True then
9:         out $\leftarrow$ (1 $\wedge$ virt_checked) $\vee$ (3 $\wedge$ virt $+ (t + s))$
10:    else // TXs belong to fundee
11:        out $\leftarrow$ (2 $\wedge$ virt_checked) $\vee$ (3 $\wedge$ virt $+ (t + s))$
12:    end if
13:    guest_all $\leftarrow$ 5 $\wedge$ $n/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1 \dots n}\}\}$
14:    guest_out $\leftarrow$ 4 $\wedge$ $2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
15:    guest $\leftarrow$ (guest_out $+ (t + s)) \vee$ guest_all
16:    $\text{TX}_{\text{base}} \leftarrow$ TX {input: ($c_{\text{tot}}$, fund), outputs: (($c_{\text{tot}} - c_{\text{guest}}$, out), ($c_{\text{guest}}$,
   guest))}
17:    return $\text{TX}_{\text{base}}$
```

Fig. 31.

Process VIRT.SIBLINGSIGS()

```
1: parse input as $\text{sigs}_{\text{byLeft}}$
2: $(\text{TX}_{\text{loc,none}}, \text{TX}_{\text{loc,left}}, \text{TX}_{\text{loc,right}}, \text{TX}_{\text{loc,both}}) \leftarrow$ VIRT.GETMIDTXS($c_{\text{guest}}, c_P, c_{\bar{P}}, c_{\text{sib}}, c_{\text{sib,rem}}, pk_{\text{sib,rem},F}, pk_{\text{sib},F}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{sib,rem,virt}}, pk_{\text{loc,virt}}, pk_{\text{loc,virt}}, pk_{\text{rem,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}, pk_{\text{loc,virt}}, \{pk_{\text{sec},i}\}_{i \in 1 \dots n}$)
3: store all signatures in $\text{sigs}_{\text{byLeft}}$ that sign any of $\text{TX}_{\text{loc,none}}$, $\text{TX}_{\text{loc,left}}$, $\text{TX}_{\text{loc,right}}$, $\text{TX}_{\text{loc,both}}$ and remove these signatures from $\text{sigs}_{\text{byLeft}}$
4: ensure that the stored signatures contain one valid signature for $\text{TX}_{\text{loc,right}}$ and $\text{TX}_{\text{loc,both}}$ which sign the guest_all input by each one of the previous $j-1$ hops
5: ensure that there are exactly 4 more valid signatures in the stored signatures, which sign the $1 \land$ left_virt_checked inputs of $\text{TX}_{\text{loc,left}}$ and $\text{TX}_{\text{loc,both}}$ with $pk_{\text{sib,rem,virt}}$ and $pk_{\text{left,guest}}$
6: $\text{sigs}_{\text{toRight}} \leftarrow \text{sigs}_{\text{byLeft}}$
7: for each hop apart from the first, the last and ours ($i \in [2, \dots, n-1] \setminus \{j\}$) do // $j$ is our hop number, hop data encoded in left_data and right_data
8:     extract data needed for GETMIDTXS() from left_data (if $i < j$) or right_data (if $i > j$) and assign it to $\text{data}_i$ and $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$ // $P$ and comm_keys are missing, that is OK. $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$ contains each party's $pk_{i,\text{virt}}$
9:     $(\text{TX}_{i,\text{none}}, \text{TX}_{i,\text{left}}, \text{TX}_{i,\text{right}}, \text{TX}_{i,\text{both}}) \leftarrow$ VIRT.GETMIDTXS($\text{data}_i$, $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$)
10:    add SIGN($\text{TX}_{i,\text{right}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) and SIGN($\text{TX}_{i,\text{both}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) to $\text{sigs}_{\text{toLeft}}$ if $i < j$, or $\text{sigs}_{\text{toRight}}$ if $i > j$ // if $i$-th hop is adjacent, 2 signatures will be produced by each SIGN() invocation: one for the guest_all and one for the $2 \land$ right_virt_checked input
11:    if $i - j = 1$ then // hop is our next
12:        add SIGN($\text{TX}_{i,\text{left}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) to $\text{sigs}_{\text{toRight}}$
13:    else if $j - i = 1$ then // hop is our previous
14:        add SIGN($\text{TX}_{i,\text{left}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) to $\text{sigs}_{\text{toLeft}}$
15:    end if
16: end for
17: if right_data does not contain data from a second-next hop then // next hop is host_fundee
18:     $\text{TX}_{\text{next,none}} \leftarrow$ VIRT.GETEDGETXS($c_{\text{guest}}, c_P, c_{\bar{P}}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{loc,virt}}, pk_{\text{rem,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}$, False)
19: end if
20: call $\bar{P}$.CIRCULATEVIRTUALSIGS($\text{sigs}_{\text{toRight}}$) and assign returned value to $\text{sigs}_{\text{byRight}}$
21: store all signatures in $\text{sigs}_{\text{byRight}}$ that sign any of $\text{TX}_{\text{loc,none}}$, $\text{TX}_{\text{loc,left}}$, $\text{TX}_{\text{loc,right}}$, $\text{TX}_{\text{loc,both}}$ and remove these signatures from $\text{sigs}_{\text{byRight}}$
22: ensure that the stored signatures contain one valid signature for $\text{TX}_{\text{loc,right}}$ and $\text{TX}_{\text{loc,both}}$ which sign the guest_all input by each one of the next $n-j$ hops
23: ensure that there are exactly 4 more valid signatures in the stored signatures, which sign the $2 \land$ right_virt_checked inputs of $\text{TX}_{\text{loc,right}}$ and $\text{TX}_{\text{loc,both}}$ with $pk_{\text{rem,virt}}$ and $pk_{\text{right,guest}}$
24: output (VIRTUALSIGSBACK, $\text{sigs}_{\text{toLeft}}$, $\text{sigs}_{\text{byRight}}$)
```

Fig. 32.

Process VIRT.INTERMEDIARYSIGS()

```
1: $(\mathrm{TX}_{\mathrm{loc,none}}, \mathrm{TX}_{\mathrm{loc,left}}, \mathrm{TX}_{\mathrm{loc,right}}, \mathrm{TX}_{\mathrm{loc,both}}) \leftarrow$ VIRT.GETMIDTXS($c_{\mathrm{guest}}, c_P, c_{\bar{P}}, c_{\mathrm{sib}}, c_{\mathrm{sib,rem}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{sib},F}, pk_{\mathrm{sib,rem}F}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{sib,rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, pk_{\mathrm{loc,virt}}, \{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n}$)
2: // not verifying our signatures in $\mathrm{sigs}_{\mathrm{byLeft}}$, our (trusted) sibling will do that
3: input (VIRTUAL SIGS FORWARD, $\mathrm{sigs}_{\mathrm{byLeft}}$) to sibling
4: VIRT.SIBLINGSIGS()
5: $\mathrm{sigs}_{\mathrm{toLeft}} \leftarrow \mathrm{sigs}_{\mathrm{byRight}} + \mathrm{sigs}_{\mathrm{toLeft}}$
6: if left_data does not contain data from a second-previous hop then // previous hop is host_funder
7:     $\mathrm{TX}_{\mathrm{prev,none}} \leftarrow$ VIRT.GETEDGETXS($c_{\mathrm{guest}}, c_{\bar{P}}, c_P, pk_{\mathrm{rem},F}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}$, True)
8: end if
9: return $\mathrm{sigs}_{\mathrm{toLeft}}$
```

Fig. 33.

Process VIRT.HOSTFUNDEESIGS()

```
1: $\mathrm{TX}_{\mathrm{loc,none}} \leftarrow$ VIRT.GETEDGETXS($c_{\mathrm{guest}}, c_P, c_{\bar{P}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}$, False)
2: for each hop apart from the first and ours ($i \in [2, \dots, n-1]$) do // hop data encoded in left_data
3:     extract data needed for GETMIDTXS() from left_data and assign it to 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 $pk_{i,\mathrm{virt}}$
4:     $(\mathrm{TX}_{i,\mathrm{none}}, \mathrm{TX}_{i,\mathrm{left}}, \mathrm{TX}_{i,\mathrm{right}}, \mathrm{TX}_{i,\mathrm{both}}) \leftarrow$ VIRT.GETMIDTXS(data$_i$, $\{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n}$)
5:     add SIGN($\mathrm{TX}_{i,\mathrm{right}}, sk_{\mathrm{loc,virt}}$, ANYPREVOUT) and SIGN($\mathrm{TX}_{i,\mathrm{both}}, sk_{\mathrm{loc,virt}}$, ANYPREVOUT) to $\mathrm{sigs}_{\mathrm{toLeft}}$ // if $i$-th hop is adjacent, 2 signatures will be produced by each SIGN() invocation: one for the guest_all and one for the $2 \wedge$ right_virt_checked input
6:     output (SIGN TXS, $\mathrm{TX}_{i,\mathrm{left}}, \mathrm{TX}_{i,\mathrm{right}}, \mathrm{TX}_{i,\mathrm{both}}$) to fundee and expect reply (TXS SIGNED, $\mathrm{sigs}_{\mathrm{guest}}$)
7:     add $\mathrm{sigs}_{\mathrm{guest}}$ to $\mathrm{sigs}_{\mathrm{toLeft}}$
8:     if $i = n - 1$ then // hop is our previous
9:         add SIGN($\mathrm{TX}_{i,\mathrm{left}}, sk_{\mathrm{loc,virt}}$, ANYPREVOUT) to $\mathrm{sigs}_{\mathrm{toLeft}}$
10:     end if
11: end for
12: return $\mathrm{sigs}_{\mathrm{toLeft}}$
```

Fig. 34.

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

Fig. 35.

```
Process virt.circulateVirtualSigs($\mathrm{sigs}_{\mathrm{byLeft}}$)
1: if $\mathrm{sigs}_{\mathrm{byLeft}}$ is given as argument then // we are not host_funder
2:     if we have a sibling then // we are not host_fundee
3:         return virt.intermediarySigs()
4:     else // we are host_fundee
5:         return virt.hostFundeeSigs()
6:     end if
7: else // we are host_funder
8:     return virt.hostFunderSigs()
9: end if
```

Fig. 36.

Process VIRT.CIRCULATEFUNDINGSIGS($\text{sig}_{\text{loc,none}}$)

```
1: if $\text{sig}_{\text{loc,none}}$ is given as argument then // we are not host_funder
2:     ensure VERIFY($\text{TX}_{\text{loc,none}}$, $\text{sig}_{\text{loc,none}}$, $pk_{\text{prev},F}$) = True // $pk_{\text{prev},F}$, found in left_data
3:     $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
4:     if we have a sibling then // we are not host_fundee
5:         input (VIRTUAL BASE SIG FORWARD, $\text{sig}_{\text{loc,none}}$) to sibling // sibling needs $\text{sig}_{\text{loc,none}}$ for closing
6:         $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
7:         $\text{sig}_{\text{next,none}} \leftarrow$ SIGN($\text{TX}_{\text{next,none}}$, $sk_{\text{loc},F}$)
8:         call VIRT.CIRCULATEVIRTUALSIGS($\text{sig}_{\text{next,none}}$) of $\bar{P}$ and assign returned value to $\text{sig}_{\text{loc,none}}$
9:         ensure VERIFY($\text{TX}_{\text{loc,none}}$, $\text{sig}_{\text{loc,none}}$, $pk_{\text{next},F}$) = True // $pk_{\text{next},F}$, found in right_data
10:        add $\text{sig}_{\text{loc,none}}$ to $\text{sigs}_{\text{loc,none}}$
11:        output (VIRTUAL BASE SIG BACK, $\text{sig}_{\text{loc,none}}$) // sibling needs $\text{sig}_{\text{loc,none}}$ for closing
12:        add $\text{sig}_{\text{loc,none}}$ to $\text{sigs}_{\text{loc,none}}$
13:    end if
14:    $\text{sig}_{\text{prev,none}} \leftarrow$ SIGN($\text{TX}_{\text{prev,none}}$, $sk_{\text{loc},F}$)
15:    return $\text{sig}_{\text{prev,none}}$
16: else // we are host_funder
17:    $\text{sig}_{\text{next,none}} \leftarrow$ SIGN($\text{TX}_{\text{next,none}}$, $sk_{\text{loc},F}$)
18:    call VIRT.CIRCULATEFUNDINGSIGS($\text{sig}_{\text{next,none}}$) of $\bar{P}$ and assign returned value to $\text{sig}_{\text{loc,none}}$
19:    ensure VERIFY($\text{TX}_{\text{loc,none}}$, $\text{sig}_{\text{loc,none}}$, $pk_{\text{next},F}$) = True // $pk_{\text{next},F}$ found in right_data
20:    $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
21:    return (OK)
22: end if
```

Fig. 37.

```
Process virt.circulateRevocations(revoc_by_prev)
1: if revoc_by_prev is given as argument then // we are not host_funder
2:     ensure guest.processRemoteRevocation(revoc_by_prev) returns (OK)
3: else // we are host_funder
4:     revoc_for_next ← guest.revokePrevious()
5:     input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to $\Sigma$
6:     last_poll ← $|\Sigma|$
7:     call virt.circulateRevocations(revoc_for_next) of $\bar{P}$ and assign
       returned value to revoc_by_next
8:     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)
9:     return (OK)
10: end if
11: if we have a sibling then // we are not host_fundee nor host_funder
12:     input (VIRTUAL REVOCATION FORWARD) to sibling
13:     revoc_for_next ← guest.revokePrevious()
14:     input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to $\Sigma$
15:     last_poll ← $|\Sigma|$
16:     call virt.circulateRevocations(revoc_for_next) of $\bar{P}$ and assign
        output to revoc_by_next
17:     ensure guest.processRemoteRevocation(revoc_by_next) returns (OK)
18:     output (HOSTS READY) to guest and expect reply (HOST ACK)
19:     output (VIRTUAL REVOCATION BACK)
20: end if
21: revoc_for_prev ← guest.revokePrevious()
22: output (HOSTS READY) to guest and expect reply (HOST ACK)
23: return revoc_for_prev // we are not host_fundee nor host_funder
```

Fig. 38.

Process VIRT – poll

```
1: On input (CHECK FOR LATERAL CLOSE) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$:
2:     input (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign output to $\Sigma$
3:     $\texttt{prev\_went\_on\_chain} \leftarrow \mathrm{TX}_{\mathrm{prev,left}} \in \Sigma \vee \mathrm{TX}_{\mathrm{prev,none}} \in \Sigma$
4:     $\texttt{next\_went\_on\_chain} \leftarrow \mathrm{TX}_{\mathrm{next,right}} \in \Sigma \vee \mathrm{TX}_{\mathrm{next,none}} \in \Sigma$
5:     $\texttt{last\_poll} \leftarrow |\Sigma|$
6:     if $\texttt{prev\_went\_on\_chain} \vee \texttt{next\_went\_on\_chain}$ then
7:         ignore all messages except for (CHECK CHAIN FOR CLOSING) by $R$
8:         $State \leftarrow$ CLOSING
9:     end if
10:    if $\texttt{prev\_went\_on\_chain} \wedge \texttt{next\_went\_on\_chain}$ then
11:        VIRT.SIGNANDSUBMIT($\mathrm{TX}_{\mathrm{loc,both}}$, $\mathrm{sigs}_{\mathrm{loc,both}}$)
12:    else if $\texttt{prev\_went\_on\_chain}$ then
13:        VIRT.SIGNANDSUBMIT($\mathrm{TX}_{\mathrm{loc,left}}$, $\mathrm{sigs}_{\mathrm{loc,left}}$)
14:    else if $\texttt{next\_went\_on\_chain}$ then
15:        VIRT.SIGNANDSUBMIT($\mathrm{TX}_{\mathrm{loc,right}}$, $\mathrm{sigs}_{\mathrm{loc,right}}$)
16:    end if

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

Fig. 39.

Process VIRT – close

```
1: On input (CLOSE) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$: // At most one of funder, fundee is defined
2:     if $State$ = CLOSED then output (CLOSED) to $R$
3:     if $State$ = GUEST PUNISHED then output (GUEST PUNISHED) to $R$
4:     ensure $State$ = OPEN
5:     if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ then // $\texttt{host}_P$ is a VIRT
6:         ignore all messages except for output (CLOSED) by $\texttt{host}_P$. Also relay to $\texttt{host}_P$ any (CHECK CHAIN FOR CLOSING) input received
7:         input (CLOSE) to $\texttt{host}_P$
8:     end if
9:     // if we have a $\texttt{host}_P$, continue from here on output (CLOSED) by it
10:    send (READ) to $\mathcal{G}_{\text{Ledger}}$ as $R$ and assign reply to $\Sigma$
11:    let tx be the unique valid TX for $\Sigma$ 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
12:    let sigs be the corresponding set of signatures among ($\text{sigs}_{\text{loc,none}}$, $\text{sigs}_{\text{loc,left}}$, $\text{sigs}_{\text{loc,right}}$, $\text{sigs}_{\text{loc,both}}$)
13:    add SIGN(tx, $sk_{A,F}$) and SIGN(tx, $sk_{\text{loc,virt}}$) to 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.
14:    ignore all messages except for (CHECK CHAIN FOR CLOSING) by $R$
15:    $State \leftarrow$ CLOSING
16:    send (SUBMIT, (tx, sigs)) to $\mathcal{G}_{\text{Ledger}}$

17: On (CHECK CHAIN FOR CLOSING) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$:
18:    ensure $State$ = CLOSING
19:    send (READ) to $\mathcal{G}_{\text{Ledger}}$ as $R$ and assign reply to $\Sigma$
20:    if $R = \texttt{guest}$ then
21:        $pk_1 \leftarrow pk_{\text{left,guest}}$; $pk_2 \leftarrow pk_{\text{right,guest}}$
22:    else // $R \in \{\texttt{funder}, \texttt{fundee}\}$
23:        $pk_1 \leftarrow pk_{\text{loc,virt}}$; $pk_2 \leftarrow pk_{\text{rem,virt}}$
24:    end if
25:    if $\Sigma$ 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
26:        $State \leftarrow$ CLOSED
27:        output (CLOSED) to $R$
28:    end if
```

Fig. 40.

Process VIRT – punishment handling

```
1: On input (USED REVOCATION) by guest: // (USED REVOCATION) by
   funder/fundee is ignored
2:     $State \leftarrow$ GUEST PUNISHED
3:     input (USED REVOCATION) to host$_P$, expect reply (USED REVOCATION OK)
4:     if funder or fundee is defined then
5:         output (ENABLER USED REVOCATION) to it
6:     else // sibling is defined
7:         output (ENABLER USED REVOCATION) to sibling
8:     end if

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

12: On output (USED REVOCATION) by host$_P$:
13:    $State \leftarrow$ GUEST PUNISHED
14:    if funder or fundee is defined then
15:        output (ENABLER USED REVOCATION) to it
16:    else // sibling is defined
17:        output (ENABLER USED REVOCATION) to sibling
18:    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* (OPEN, $c, \dots$) *by either $\mathcal{E}$ or $\bar{P}$,*
- *P [has received* (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* (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* host$_P \neq \mathcal{G}_{\text{Ledger}}$ *the output of* 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*

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

with overwhelming probability in the security parameter.

Proof. Define the *history* of a channel as follows: $H = (F, C)$, 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, ...) 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 l. 21 or l. 6 of Fig. 20 respectively is executed by P; this represents the balance change after each payment. Moving on to the funding of new virtual channels, whenever P executes l. 20 of Fig. 17 (therefore funding a new virtual channel), $[-c_{\text{guest}}]$ is appended to C and whenever l. 23 of Fig. 17 is executed (in which case P helps with the opening of a new virutal channel, but does not fund it), $[0]$ 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.

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).

Before transitioning to the OPEN *State*, P has produced only one valid signature for the 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. □

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 $(\text{OPEN}, c, \dots)$ by $\mathcal{E}$ prior to transitioning to the OPEN State,*
- *P [has received $(\text{FUND ME}, f_i, \dots)$ 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 $(\text{GET PAID}, e_i)$ 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 $(\text{FUND ME}, f_i, \dots)$ 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 ($\textsc{get paid}, e_i$) by $\mathcal{E}$ while $State_P = \textsc{open}$, e_i is stored (Fig. 3, l. 7). The next time $State_P$ transitions to $\textsc{open}$ (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 $\sum\limits_{i=1}^{l} e_i$ in total.

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

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 $\textsc{ln}$ and the same $\mathcal{E}$ and $\mathcal{A}$ hold as well for the same m, n and l values.

For $State_P$ to become $\textsc{ignored}$, either $\mathcal{S}$ has to send ($\textsc{became corrupted or negligent}$, P) or $\texttt{host}_P$ must output ($\textsc{enabler used revocation}$) to $\mathcal{F}_{\text{Chan}}$ (Fig. 2, l. 4). The first case only happens when either P receives ($\textsc{corrupt}$) 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 $\textsc{guest punished}$ (Fig. 41, ll. 1 or 12), so in case P receives ($\textsc{close}$) by $\mathcal{E}$ the output of $\texttt{host}_P$ will be ($\textsc{guest punished}$) (Fig. 40, l. 3). In all cases, some condition of Lemma 1 is violated.

For $State_P$ to become $\textsc{open}$ at least once, the following sequence of events must take place (Fig. 2): If $P = Alice$, it must receive ($\textsc{init}$, pk) by $\mathcal{E}$ when $State_P = \textsc{uninit}$, then either receive ($\textsc{open}$, c, $\mathcal{G}_{\text{Ledger}}$, $\ldots$) by $\mathcal{E}$ and ($\textsc{base open}$) by $\mathcal{S}$ or ($\textsc{open}$, c, $\texttt{hops}$ ($\neq \mathcal{G}_{\text{Ledger}}$), $\ldots$) by $\mathcal{E}$, ($\textsc{funded}$, $\textsc{host}$, $\ldots$) by $\texttt{hops}[0].\texttt{left}$ and ($\textsc{virtual open}$) by $\mathcal{S}$. In either case, $\mathcal{S}$ only sends its message only if all its simulated honest parties move to the $\textsc{open}$ 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 $\textsc{open}$ for the first time only if all honest simulated parties move to the $\textsc{open}$ *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 $\textsc{open}$ 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 $\textsc{open}$, then the state of the other is also $\textsc{open}$. Each time P receives input ($\textsc{fund me}$, 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 $\mathit{State}_P = \text{IGNORED}$ 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 $\mathit{State}_P \neq \text{IGNORED}$ (Fig. 5, l. 3). We deduce that, once $\mathit{State}_P = \text{IGNORED}$, 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 $P = \mathit{Alice}$, 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,*

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

Proof. By inspection of Figs. 1 and 6 we can deduce that for a particular $\mathcal{E}$, in the ideal world execution $\text{EXEC}^{\mathcal{F}_{\text{Chan}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S}_{\mathcal{A}},\mathcal{E}}$, $\mathcal{S}_{\mathcal{A}}$ simulates internally the two LN parties exactly as they would execute in the real world execution, $\text{EXEC}^{\mathcal{G}_{\text{Ledger}}}_{\text{LN},\mathcal{A},\mathcal{E}}$ 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. □

References