

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 $\text{mode} \in \{\text{input}, \text{output}, \text{network}\}$, handle it according to the corresponding rule in Fig. 2, 3, 5, or 4 (if any) and subsequently send $(\text{RELAY}, \text{msg}, P, \mathcal{E}, \text{input})$ \mathcal{A} .
// all messages are relayed to \mathcal{A}
- On receiving $(\text{RELAY}, \text{msg}, P, R, \text{mode})$ by \mathcal{A} ($\text{mode} \in \{\text{input}, \text{output}, \text{network}\}$, $P \in \{Alice, Bob\}$), relay msg to R as P by means of mode . // \mathcal{A} fully controls outgoing messages by $\mathcal{F}_{\text{Chan}}$
- On receiving $(\text{INFO}, \text{msg})$ 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 $(\text{HANDLED}, \text{msg})$ to \mathcal{A} . // $(\text{INFO}, \text{msg})$ 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 $P \in \{\text{Alice}, \text{Bob}\}$

- 1: On first activation: // before handing the message
- 2: $pk_P \leftarrow \perp$; $\text{host}_P \leftarrow \perp$; $\text{enabler}_P \leftarrow \perp$; $\text{balance}_P \leftarrow 0$;
- 3: $\text{State}_P \leftarrow \text{UNINIT}$
- 4: On (BECAME CORRUPTED OR NEGLIGENT, P) by \mathcal{A} or on output (ENABLER USED REVOCATION) by host_P when in any state:
- 5: $\text{State}_P \leftarrow \text{IGNORED}$
- 6: On (INIT, pk) to P by \mathcal{E} when $\text{State}_P = \text{UNINIT}$:
- 7: $pk_P \leftarrow pk$
- 8: $\text{State}_P \leftarrow \text{INIT}$
- 9: On (OPEN, x , $\mathcal{G}_{\text{Ledger}}, \dots$) to *Alice* by \mathcal{E} when $\text{State}_A = \text{INIT}$:
- 10: store x
- 11: $\text{State}_A \leftarrow \text{TENTATIVE BASE OPEN}$
- 12: On (BASE OPEN) by \mathcal{A} when $\text{State}_A = \text{TENTATIVE BASE OPEN}$:
- 13: $\text{balance}_A \leftarrow x$
- 14: $\text{State}_A \leftarrow \text{OPEN}$
- 15: On (BASE OPEN) by \mathcal{A} when $\text{State}_B = \text{INIT}$:
- 16: $\text{State}_B \leftarrow \text{OPEN}$
- 17: On (OPEN, x , $\text{hops} \neq \mathcal{G}_{\text{Ledger}}, \dots$) to *Alice* by \mathcal{E} when $\text{State}_A = \text{INIT}$:
- 18: store x
- 19: $\text{enabler}_A \leftarrow \text{hops}[0].\text{left}$
- 20: $\text{State}_A \leftarrow \text{PENDING VIRTUAL OPEN}$
- 21: On output (FUNDED, host , \dots) to *Alice* by enabler_A when $\text{State}_A = \text{PENDING VIRTUAL OPEN}$:
- 22: $\text{host}_A \leftarrow \text{host}[0].\text{left}$
- 23: $\text{State}_A \leftarrow \text{TENTATIVE VIRTUAL OPEN}$
- 24: On output (FUNDED, host , \dots) to *Bob* by ITI $R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$ when $\text{State}_B = \text{INIT}$:
- 25: $\text{enabler}_B \leftarrow R$
- 26: $\text{host}_B \leftarrow \text{host}$
- 27: $\text{State}_B \leftarrow \text{TENTATIVE VIRTUAL OPEN}$
- 28: On (VIRTUAL OPEN) by \mathcal{A} when $\text{State}_P = \text{TENTATIVE VIRTUAL OPEN}$:
- 29: **if** $P = \text{Alice}$ **then** $\text{balance}_P \leftarrow x$
- 30: $\text{State}_P \leftarrow \text{OPEN}$

Fig. 2.

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

```

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

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

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

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

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

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

```

Fig. 3.

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

```

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

4: When  $\text{State}_P = \text{PENDING FUND}$ :
5:   if we intercept the command “define new VIRT ITI host” by  $\mathcal{A}$ , routed
   through  $P$  then
6:     store host
7:      $\text{State}_P \leftarrow \text{TENTATIVE FUND}$ 
8:     continue executing  $\mathcal{A}$ ’s command
9:   end if

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

12: When  $\text{State}_P = \text{OPEN}$ :
13:   if we intercept the command “define new VIRT ITI host” by  $\mathcal{A}$ , routed
   through  $P$  then
14:     store host
15:      $\text{State}_P \leftarrow \text{TENTATIVE HELP FUND}$ 
16:     continue executing  $\mathcal{A}$ ’s command
17:   end if

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

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

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

```

Fig. 4.

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

- 1: On (CLOSE) by \mathcal{E} when $\text{State}_P = \text{OPEN}$:
- 2: $\text{State}_P \leftarrow \text{CLOSING}$
- 3: On (CLOSE, P) by \mathcal{A} when $\text{State} \notin \{\text{UNINIT}, \text{INIT}, \text{PENDING VIRTUAL OPEN}, \text{TENTATIVE VIRTUAL OPEN}, \text{TENTATIVE BASE OPEN}, \text{IGNORED}\}$:
- 4: input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign output to Σ
- 5: $\text{coins}_P \leftarrow$ sum of coins exclusively spendable by pk_P in Σ
- 6: **if** $\text{coins}_P \geq \text{balance}_P$ **then**
- 7: $\text{State}_P \leftarrow \text{CLOSED}$
- 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 $\mathcal{F}_{\text{Chan}}$ ($\text{in_mode} \in \{\text{input}, \text{output}, \text{network}\}$, $P \in \{\text{Alice}, \text{Bob}\}$), 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 $\text{out_mode} \in \{\text{input}, \text{output}, \text{network}\}$, 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} , addressed 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 20 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. 20
 - 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, 21): 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/Fig. 27, l. 27):
 - 21: send (INFO, CLOSE, P) to $\mathcal{F}_{\text{Chan}}$

Fig. 7.

Process LN – init

```

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

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

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

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

```

Fig. 8.

Process LN – methods used by VIRT

```

1: REVOKEPREVIOUS():
2:   ensure  $State \in \text{WAITING FOR (OUTBOUND) REVOCATION}$ 
3:    $R_{\bar{P},i} \leftarrow \text{TX}$  {input:  $C_{P,i}.\text{outputs}.P$ , output:  $(C_{P,i}.\text{outputs}.P.\text{value},$ 
       $pk_{\bar{P},\text{out}})$ }
4:    $\text{sig}_{A,R,i} \leftarrow \text{SIGN}(R_{\bar{P},i}, sk_{P,R})$ 
5:   if  $State = \text{WAITING FOR REVOCATION}$  then
6:      $State \leftarrow \text{WAITING FOR INBOUND REVOCATION}$ 
7:   else //  $State = \text{WAITING FOR OUTBOUND REVOCATION}$ 
8:      $i \leftarrow i + 1$ 
9:      $State \leftarrow \text{WAITING FOR HOSTS READY}$ 
10:  end if
11:   $\text{host}_P \leftarrow \text{host}'_P$  // forget old host, use new host instead
12:   $\text{layer} \leftarrow \text{layer} + 1$ 
13:  return  $\text{sig}_{P,R,i}$ 

14: PROCESSREMOTEREVOCATION( $\text{sig}_{\bar{P},R,i}$ ):
15:   ensure  $State = \text{WAITING FOR (INBOUND) REVOCATION}$ 
16:    $R_{P,i} \leftarrow \text{TX}$  {input:  $C_{\bar{P},i}.\text{outputs}.P$ , output:  $(C_{\bar{P},i}.\text{outputs}.\bar{P}.\text{value},$ 
       $pk_{P,\text{out}})$ }
17:   ensure  $\text{VERIFY}(R_{P,i}, \text{sig}_{\bar{P},R,i}, pk_{\bar{P},R}) = \text{True}$ 
18:   if  $State = \text{WAITING FOR REVOCATION}$  then
19:      $State \leftarrow \text{WAITING FOR OUTBOUND REVOCATION}$ 
20:   else //  $State = \text{WAITING FOR INBOUND REVOCATION}$ 
21:      $i \leftarrow i + 1$ 
22:      $State \leftarrow \text{WAITING FOR HOSTS READY}$ 
23:   end if
24:   return (OK)

25: NEGLIGENT():
26:    $\text{negligent} \leftarrow \text{True}$ 
27:   return (OK)

```

Fig. 9.

Process LN.EXCHANGEOPENKEYS()

```

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

```

Fig. 10.

Process LN.PREPAREBASE()

```

1: if hops =  $\mathcal{G}_{\text{Ledger}}$  then // opening base channel
2:    $F \leftarrow \text{TX} \{\text{input: } (c, pk_{A,\text{chain}}), \text{output: } (c, 3 \wedge 2 / \{pk_{A,F}, pk_{B,F}\})\}$ 
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  $\leftarrow \text{funder\_layer}$ 
8: end if

```

Fig. 11.

Process LN.EXCHANGEOPENSIGS()

```

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

```

Fig. 12.

Process LN.COMMITBASE()

```

1:  $\text{sig}_F \leftarrow \text{SIGN}(F, sk_{A,\text{chain}})$ 
2: send (OPEN,  $c, pk_{A,\text{out}}, pk_{B,\text{out}}, F, \text{sig}_F, \text{Alice}, \text{Bob}$ ) 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  $State = \text{WAITING TO CHECK FUNDING}$ 
3:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
4:   if  $F \in \Sigma$  then
5:      $State \leftarrow \text{OPEN}$ 
6:     reply (OPEN OK)
7:   end if

8: On output (FUNDED,  $host_P$ ,  $funder\_layer$ ) by  $hops[-1].right$ :
9:   ensure  $State = \text{WAITING FOR FUNDED}$ 
10:  store  $host_P$  // we will talk directly to  $host_P$ 
11:   $layer \leftarrow funder\_layer$ 
12:   $State \leftarrow \text{WAITING FOR COMM SIG}$ 
13:  reply (FUND ACK)

14: On output (CHECK KEYS,  $(pk_1, pk_2)$ ) by  $hops[-1].right$ :
15:  ensure  $State = \text{WAITING FOR CHECK KEYS}$ 
16:  ensure  $pk_1 = pk_{A,F} \wedge pk_2 = pk_{B,F}$ 
17:   $State \leftarrow \text{WAITING FOR FUNDED}$ 
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:  $\text{sig}_{P,C,i+1} \leftarrow \text{SIGN}(C_{\bar{P},i+1})$  // 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:  $\text{sig}_{\bar{P},C,i+1} \leftarrow \text{SIGN}(C_{P,i+1}, sk'_{\bar{P},F})$  // 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  $State = \text{OPEN}$ 
3:   ensure  $c_P - \text{locked}_P \geq c_{\text{guest}}$ 
4:    $State \leftarrow \text{VIRTUALISING}$ 
5:    $(sk'_{P,F}, pk'_{P,F}) \leftarrow \text{KEYGEN}()$ 
6:   define new VIRT ITI  $\text{host}'_P$ 
7:   send (VIRTUALISING,  $\text{host}'_P$ ,  $pk'_{P,F}$ , hops, fundee,  $c_{\text{guest}}$ ) to  $\bar{P}$  and expect
      reply (VIRTUALISING ACK,  $\text{host}'_{\bar{P}}$ ,  $pk'_{\bar{P},F}$ )
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:   $State \leftarrow \text{WAITING FOR REVOCATION}$ 
11:  input (HOST ME, funder, fundee,  $\text{host}'_{\bar{P}}$ ,  $\text{host}_P$ ,  $c_{\text{guest}}$ ,  $pk_{A,V}$ ,  $pk_{B,V}$ ,
       $(sk'_{P,F}, pk'_{P,F})$ ,  $(sk_{P,F}, pk_{P,F})$ ,  $pk_{\bar{P},F}$ ,  $pk'_{\bar{P},F}$ ) to  $\text{host}'_P$ 

12: On output (HOSTS READY) by  $\text{host}_P$ : //  $\text{host}_P$  is the new host, renamed in
      Fig. 9, l. 12
13:   ensure  $State = \text{WAITING FOR HOSTS READY}$ 
14:    $State \leftarrow \text{OPEN}$ 
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  $\text{len}(\text{hops}) = 1$  then // we are the last hop
18:     output (FUNDED,  $\text{host}_P$ , 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,  $\text{host}_P$ , layer) to funder // do not expect reply by
      funder
22:   end if
23:   reply (HOST ACK)

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

17: On input (VIRTUALISING,  $\text{host\_sibling}, (sk'_{P,F}, pk'_{P,F}), pk_{\text{sib},\bar{P},F}, \text{hops},$ 
    $\text{fundee}, c_{\text{guest}}, c_{\text{sib},\text{rem}}, \text{sib}$ ) by sibling:
18:   ensure  $\text{State} = \text{OPEN}$ 
19:   ensure  $c_P - \text{locked}_P \geq c$ 
20:   ensure  $c_{\text{sib},\text{rem}} \geq c_P \wedge c_{\bar{P}} \geq c_{\text{sib}}$  // avoid value loss by griefing attack: one
   counterparty closes with old version, the other stays idle forever
21:    $\text{State} \leftarrow \text{VIRTUALISING}$ 
22:    $\text{locked}_P \leftarrow \text{locked}_P + c$ 
23:   define new VIRT ITI  $\text{host}'_P$ 
24:   send (VIRTUALISING,  $\text{host}'_P, pk'_{P,F}, \text{hops}, \text{fundee}, c_{\text{guest}}$ ) to  $\text{hops}[0].\text{right}$ 
   and expect reply (VIRTUALISING ACK,  $\text{host}'_{\bar{P}}, pk'_{\bar{P},F}$ )
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,  $\text{host}_P, \text{host}'_{\bar{P}}, \text{host\_sibling}, (sk'_{P,F}, pk'_{P,F}), pk'_{\bar{P},F}, pk_{\text{sib},\bar{P},F},$ 
    $(sk_{P,F}, pk_{P,F}), pk_{\bar{P},F}, c_{\text{guest}}$ ) to  $\text{host}'_P$  and expect reply (HOST INIT OK)
28:    $\text{State} \leftarrow \text{WAITING FOR REVOCATION}$ 
29:   output (VIRTUALISING ACK,  $\text{host}'_P, pk'_{P,F}$ ) 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: send (PAY,  $x$ ,  $\text{sig}_{P,C,i+1}$ ) to  $\bar{P}$ 
4: //  $P$  refers to payer and  $\bar{P}$  to payee both in local and remote code
5: ensure  $State = \text{WAITING TO GET PAID} \wedge x = y$ 
6: if  $\text{host}_{\bar{P}} \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$  has a host_sibling then // we are intermediary
    channel
7:     ensure  $c_{\text{sib,rem}} \geq c_P - x \wedge c_{\bar{P}} + x \geq c_{\text{sib}}$  // avoid value loss by griefing attack
8: end if
9:  $C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$  with  $x$  coins moved from  $P$ 's to  $\bar{P}$ 's output
10: ensure  $\text{VERIFY}(C_{\bar{P},i+1}, \text{sig}_{P,C,i+1}, pk_{P,F}) = \text{True}$ 
11:  $C_{P,i+1} \leftarrow C_{P,i}$  with  $x$  coins moved from  $P$ 's to  $\bar{P}$ 's output
12:  $\text{sig}_{\bar{P},C,i+1} \leftarrow \text{SIGN}(C_{P,i+1}, sk_{\bar{P},F})$  // kept by  $P$ 
13:  $R_{P,i} \leftarrow \text{TX}$  {input:  $C_{\bar{P},i}.\text{outputs}.P$ , output:  $(c_{\bar{P}}, pk_{P,\text{out}})$ }
14:  $\text{sig}_{\bar{P},R,i} \leftarrow \text{SIGN}(R_{P,i}, sk_{\bar{P},R})$ 
15:  $State \leftarrow \text{WAITING FOR PAY REVOCATION}$ 
16: reply (COMMITMENT SIGNED,  $\text{sig}_{\bar{P},C,i+1}$ ,  $\text{sig}_{\bar{P},R,i}$ )
17:  $C_{P,i+1} \leftarrow C_{P,i}$  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:  $R_{P,i} \leftarrow \text{TX}$  {input:  $C_{\bar{P},i}.\text{outputs}.P$ , 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:  $R_{\bar{P},i} \leftarrow \text{TX}$  {input:  $C_{P,i}.\text{outputs}.\bar{P}$ , output: ( $c_P$ ,  $pk_{\bar{P},\text{out}}$ )}
5:  $\text{sig}_{P,R,i} \leftarrow \text{SIGN}(R_{\bar{P},i}, sk_{P,R})$ 
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: if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge$  we have a host_sibling then // we are intermediary
   channel
9:   input (NEW BALANCE,  $c_P$ ,  $c_{\bar{P}}$ ) to hostP
10:  relay message as input to sibling // run by VIRT
11:  relay message as output to guest // run by VIRT
12:  store new sibling balance and reply (NEW BALANCE OK)
13:  output (NEW BALANCE OK) to sibling // run by VIRT
14:  output (NEW BALANCE OK) to guest // run by VIRT
15: end if
16: send (REVOKE AND ACK,  $\text{sig}_{P,R,i}$ ) to  $\bar{P}$ 
17: ensure State = WAITING FOR PAY REVOCATION
18:  $R_{\bar{P},i} \leftarrow \text{TX}$  {input:  $C_{P,i}.\text{outputs}.\bar{P}$ , output: ( $c_P$ ,  $pk_{\bar{P},\text{out}}$ )}
19: ensure VERIFY( $R_{\bar{P},i}$ ,  $\text{sig}_{P,R,i}$ ,  $pk_{P,R}$ ) = True
20: add  $x$  to paid_in
21:  $c_P \leftarrow c_P - x$ ;  $c_{\bar{P}} \leftarrow c_{\bar{P}} + x$ ;  $i \leftarrow i + 1$ 
22: State  $\leftarrow$  OPEN
23: if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$  has a host_sibling then // we are intermediary
   channel
24:  input (NEW BALANCE,  $c_{\bar{P}}$ ,  $c_P$ ) to host\bar{P}}
25:  relay message as input to sibling // run by VIRT
26:  relay message as output to guest // run by VIRT
27:  store new sibling balance and reply (NEW BALANCE OK)
28:  output (NEW BALANCE OK) to sibling // run by VIRT
29:  output (NEW BALANCE OK) to guest // run by VIRT
30: end if

```

Fig. 20.

Process LN – On (PAY, x):

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

Fig. 21.

Process LN – On (GET PAID, y):

- 1: ensure $State = \text{OPEN} \wedge c_{\bar{P}} \geq x$
- 2: store y
- 3: $State \leftarrow \text{WAITING TO GET PAID}$

Fig. 22.

Process LN – On (CHECK FOR LATERAL CLOSE):

- 1: **if** $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then**
- 2: input (CHECK FOR LATERAL CLOSE) to host_P
- 3: **end if**

Fig. 23.

Process LN – On (CHECK CHAIN FOR OLD COMM):

```

1: ensure  $State \notin \{\perp, \text{INIT}, \text{TOPPED UP}\}$  // 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:  $\text{last\_poll} \leftarrow |\Sigma|$ 
5: if  $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$  then // counterparty has closed maliciously
6:    $State \leftarrow \text{CLOSING}$ 
7:   LN.SUBMITANDCHECKREVOCATION( $j$ )
8:    $State \leftarrow \text{CLOSED}$ 
9: end if

```

Fig. 24.

Process LN.SUBMITANDCHECKREVOCATION(j)

```

1:  $\text{sig}_{P,R,j} \leftarrow \text{SIGN}(R_{P,j}, sk_{P,R})$ 
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  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$  then
9:   input (USED REVOCATION) to  $\text{host}_P$ 
10: end if

```

Fig. 25.

Process LN – On (CLOSE):

```

1: ensure  $State \notin \{\perp, \text{INIT}, \text{TOPPED UP}, \text{CLOSED}, \text{BASE PUNISHED}\}$  // channel open
2: if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$  then // we have a virtual channel
3:    $State \leftarrow \text{HOST CLOSING}$ 
4:   input (CLOSE) to  $\text{host}_P$  and keep relaying inputs (CHECK CHAIN FOR
   CLOSING) to  $\text{host}_P$  until receiving output (CLOSED) by  $\text{host}_P$ 
5:    $\text{host}_P \leftarrow \mathcal{G}_{\text{Ledger}}$ 
6: end if
7:  $State \leftarrow \text{CLOSING}$ 
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:  // provably reachable – TODO: ref
19:   $\text{sig}'_{P,C,i} \leftarrow \text{SIGN}(C_{P,i}, sk_{P,F})$ 
20:  input (SUBMIT,  $(C_{P,i}, \text{sig}_{P,C,i}, \text{sig}'_{P,C,i})$ ) to  $\mathcal{G}_{\text{Ledger}}$ 
21:  while  $C_{P,i} \notin \Sigma$  do
22:    wait for input (CHECK CLOSED) // ignore other messages
23:    input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
24:  end while
25:  // provably reachable – TODO: ref
26: end if
27:  $State \leftarrow \text{CLOSED}$ 
28: output (CLOSED)

```

Fig. 26.

Process LN – On output (ENABLER USED REVOCATION) by host_P :

```

1:  $State \leftarrow \text{BASE PUNISHED}$ 

```

Fig. 27.

Process VIRT

```

1: On every activation, before handling the message:
2:   if last_poll  $\neq \perp$  then // virtual layer is ready
3:     input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
4:     if last_poll +  $t < |\Sigma|$  then
5:       for  $P \in \{\text{guest}, \text{funder}, \text{fundee}\}$  do // at most 1 of funder, fundee
        is defined
6:         ensure  $P.\text{NEGLIGENT}()$  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,  $\text{host}_P$ ,  $\bar{P}$ , sibling, fundee,  $(sk_{\text{loc}, \text{virt}}, pk_{\text{loc}, \text{virt}})$ ,  $pk_{\text{rem}, \text{virt}}$ ,
     $pk_{\text{sib}, \text{rem}, \text{virt}}$ ,  $(sk_{\text{loc}, F}, pk_{\text{loc}, F})$ ,  $pk_{\text{rem}, F}$ ,  $c_{\text{guest}}$ ) 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  $\leftarrow \perp$ 
14:   output (HOST INIT OK) to guest

15: On input (HOST ME, funder, fundee,  $\bar{P}$ ,  $\text{host}_P$ ,  $c_{\text{guest}}$ ,  $pk_{\text{left}, \text{guest}}$ ,  $pk_{\text{right}, \text{guest}}$ ,
     $(sk_{\text{loc}, \text{virt}}, pk_{\text{loc}, \text{virt}})$ ,  $(sk_{\text{loc}, F}, pk_{\text{loc}, F})$ ,  $pk_{\text{rem}, F}$ ,  $pk_{\text{rem}, \text{virt}}$ ) by guest:
16:   last_poll  $\leftarrow \perp$ 
17:   ensure  $\text{VIRT.CIRCULATEKEYSANDCOINS}()$  returns (OK)
18:   ensure  $\text{VIRT.CIRCULATEVIRTUALSIGS}()$  returns (OK)
19:   ensure  $\text{VIRT.CIRCULATEFUNDINGSIGS}()$  returns (OK)
20:   ensure  $\text{VIRT.CIRCULATEREVOCATIONS}()$  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, ( $sk_{loc,virt}$ ,  $pk_{loc,virt}$ ),
      ( $sk_{loc,F}$ ,  $pk_{loc,F}$ ),  $pk_{rem,F}$ ,  $c_P$ ,  $c_{\bar{P}}$ ) to sibling
4:     store input as left_data
5:     parse left_data as far_left_data, ( $sk_{loc,virt}$ ,  $pk_{loc,virt}$ ), ( $sk_{sib,F}$ ,
       $pk_{sib,F}$ ),  $pk_{sib,rem,F}$ ,  $c_{sib}$ ,  $c_{sib,rem}$  // 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,  $pk_{rem,virt}$ 
8:     output (KEYS AND COINS BACK, right_data, ( $sk_{loc,F}$ ,  $pk_{loc,F}$ ),  $pk_{rem,F}$ ,
       $c_P$ ,  $c_{\bar{P}}$ )
9:     store output as right_data
10:    parse right_data as far_right_data, ( $sk_{sib,F}$ ,  $pk_{sib,F}$ ),  $pk_{sib,rem,F}$ ,  $c_{sib}$ ,
       $c_{sib,rem}$ 
11:    return (right_data,  $pk_{loc,virt}$ )
12:  else // we are host_fundee
13:    extract ( $pk_{left,guest}$ ,  $pk_{right,guest}$ ) from left_data
14:    output (CHECK KEYS, ( $pk_{left,guest}$ ,  $pk_{right,guest}$ )) to fundee and expect
      reply (KEYS OK)
15:    return  $pk_{loc,virt}$ 
16:  end if
17: else // we are host_funder
18:  call VIRT.CIRCULATEKEYSANDCOINS( $pk_{loc,virt}$ , ( $pk_{left,guest}$ ,  $pk_{right,guest}$ )) 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:    $\text{left\_fund} \leftarrow 2 / \{pk_{\text{left,fund}}, pk_{\text{loc,fund}}\}$ 
5:    $\text{right\_fund} \leftarrow 2 / \{pk_{\text{sib,fund}}, pk_{\text{right,fund}}\}$ 
6:    $\text{left\_virt} \leftarrow 2 / \{pk_{\text{left,virt}}, pk_{\text{loc,virt}}\}$ 
7:    $\text{left\_virt\_checked} \leftarrow 4 / \{pk_{\text{left,virt}}, pk_{\text{loc,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$ 
8:    $\text{right\_virt} \leftarrow 2 / \{pk_{\text{sib,virt}}, pk_{\text{right,virt}}\}$ 
9:    $\text{right\_virt\_checked} \leftarrow 4 / \{pk_{\text{sib,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$ 
10:   $\text{left\_out\_checked} \leftarrow (2 \wedge \text{left\_virt\_checked}) \vee (3 \wedge \text{left\_virt} + (t + s))$ 
11:   $\text{right\_out} \leftarrow (1 \wedge \text{right\_virt}) \vee (3 \wedge \text{right\_virt} + (t + s))$ 
12:
13:   $\text{right\_out\_checked} \leftarrow (1 \wedge \text{right\_virt\_checked}) \vee (3 \wedge \text{right\_virt} + (t + s))$ 
14:   $\text{guest\_all} \leftarrow 5 \wedge n / \{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1 \dots n}\}\}$ 
15:   $\text{guest\_out} \leftarrow 4 \wedge 2 / \{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$ 
16:   $\text{guest} \leftarrow (\text{guest\_out} + (t + s)) \vee \text{guest\_all}$ 
17:   $\text{TX}_{\text{none}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}}, \text{left\_fund}), (c_{\text{right}}, \text{right\_fund})), \text{outputs: } ((c_{\text{left}} - c_{\text{guest}}, \text{left\_out\_checked}), (c_{\text{right}} - c_{\text{guest}}, \text{right\_out\_checked}), (c_{\text{guest}}, pk_{\text{loc,out}}), (c_{\text{guest}}, \text{guest})) \}$ 
18:   $\text{TX}_{\text{left}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}} - c_{\text{guest}}, 1 \wedge \text{left\_virt\_checked}), (c_{\text{right}}, \text{right\_fund})), \text{outputs: } ((c_{\text{left}} - c_{\text{guest}}, 3 \wedge \text{left\_virt}), (c_{\text{right}} - c_{\text{guest}}, \text{right\_out\_checked}), (c_{\text{guest}}, pk_{\text{loc,out}})) \}$ 
19:   $\text{TX}_{\text{right}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}}, \text{left\_fund}), (c_{\text{right}} - c_{\text{guest}}, 2 \wedge \text{right\_virt\_checked}), (c_{\text{guest}}, \text{guest\_all})), \text{outputs: } ((c_{\text{left}} - c_{\text{guest}}, \text{left\_out\_checked}), (c_{\text{right}} - c_{\text{guest}}, 3 \wedge \text{right\_virt}), (c_{\text{guest}}, pk_{\text{loc,out}}), (c_{\text{guest}}, \text{guest})) \}$ 
20:   $\text{TX}_{\text{both}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}} - c_{\text{guest}}, 1 \wedge \text{left\_virt\_checked}), (c_{\text{right}} - c_{\text{guest}}, 2 \wedge \text{right\_virt\_checked}), (c_{\text{guest}}, \text{guest\_all})), \text{outputs: } ((c_{\text{left}} - c_{\text{guest}}, 3 \wedge \text{left\_virt}), (c_{\text{right}} - c_{\text{guest}}, 3 \wedge \text{right\_virt}), (c_{\text{guest}}, pk_{\text{loc,out}})) \}$ 
21:  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( $c_{\text{guest}}, c_{\text{left}}, c_{\text{right}}, pk_{\text{left},\text{fund}}, pk_{\text{right},\text{fund}}, pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}},$ 
    $pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}, \{pk_{\text{sec},i}\}_{i \in 1 \dots n}, \text{is\_funder}$ ):
3:   ensure  $c_{\text{left}} \geq c_{\text{guest}}$ 
4:    $c_{\text{tot}} \leftarrow c_{\text{left}} + c_{\text{right}}$ 
5:    $\text{fund} \leftarrow 2/\{pk_{\text{left},\text{fund}}, pk_{\text{right},\text{fund}}\}$ 
6:    $\text{virt} \leftarrow 2/\{pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}}\}$ 
7:    $\text{virt\_checked} \leftarrow 4/\{pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}}, pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}\}$ 
8:   if  $\text{is\_funder} = \text{True}$  then
9:      $\text{out} \leftarrow (1 \wedge \text{virt\_checked}) \vee (3 \wedge \text{virt} + (t + s))$ 
10:  else // TXs belong to fundee
11:     $\text{out} \leftarrow (2 \wedge \text{virt\_checked}) \vee (3 \wedge \text{virt} + (t + s))$ 
12:  end if
13:   $\text{guest\_all} \leftarrow 5 \wedge n/\{pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}, \{pk_{\text{sec},1 \dots n}\}\}$ 
14:   $\text{guest\_out} \leftarrow 4 \wedge 2/\{pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}\}$ 
15:   $\text{guest} \leftarrow (\text{guest\_out} + (t + s)) \vee \text{guest\_all}$ 
16:   $\text{TX}_{\text{base}} \leftarrow \text{TX} \{\text{input: } (c_{\text{tot}}, \text{fund}), \text{outputs: } ((c_{\text{tot}} - c_{\text{guest}}, \text{out}), (c_{\text{guest}},$ 
     $\text{guest}))\}$ 
17:  return  $\text{TX}_{\text{base}}$ 

```

Fig. 31.

Process VIRT.SIBLINGSIGS()

- 1: parse input as $\text{sigs}_{\text{byLeft}}$
- 2: $(\text{TX}_{\text{loc},\text{none}}, \text{TX}_{\text{loc},\text{left}}, \text{TX}_{\text{loc},\text{right}}, \text{TX}_{\text{loc},\text{both}}) \leftarrow \text{VIRT.GETMIDTXS}(c_{\text{guest}}, c_P, c_{\bar{P}}, c_{\text{sib}}, c_{\text{sib},\text{rem}}, pk_{\text{sib},\text{rem},F}, pk_{\text{sib},F}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{sib},\text{rem},\text{virt}}, pk_{\text{loc},\text{virt}}, pk_{\text{rem},\text{virt}}, pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}, pk_{\text{loc},\text{virt}}, \{pk_{\text{sec},i}\}_{i \in 1 \dots n})$
- 3: store all signatures in $\text{sigs}_{\text{byLeft}}$ that sign any of $\text{TX}_{\text{loc},\text{none}}, \text{TX}_{\text{loc},\text{left}}, \text{TX}_{\text{loc},\text{right}}, \text{TX}_{\text{loc},\text{both}}$ and remove these signatures from $\text{sigs}_{\text{byLeft}}$
- 4: ensure that the stored signatures contain one valid signature for $\text{TX}_{\text{loc},\text{right}}$ and $\text{TX}_{\text{loc},\text{both}}$ 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 $1 \wedge \text{left_virt_checked}$ inputs of $\text{TX}_{\text{loc},\text{left}}$ and $\text{TX}_{\text{loc},\text{both}}$ with $pk_{\text{sib},\text{rem},\text{virt}}$ and $pk_{\text{left},\text{guest}}$
- 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 $\text{GETMIDTXS}()$ from **left_data** (if $i < j$) or **right_data** (if $i > j$) and assign it to 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: $(\text{TX}_{i,\text{none}}, \text{TX}_{i,\text{left}}, \text{TX}_{i,\text{right}}, \text{TX}_{i,\text{both}}) \leftarrow \text{VIRT.GETMIDTXS}(\text{data}_i, \{pk_{\text{sec},i}\}_{i \in 1 \dots n})$
- 10: add $\text{SIGN}(\text{TX}_{i,\text{right}}, sk_{\text{loc},\text{virt}}, \text{ANYPREVOUT})$ and $\text{SIGN}(\text{TX}_{i,\text{both}}, sk_{\text{loc},\text{virt}}, \text{ANYPREVOUT})$ 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 $\text{SIGN}()$ invocation: one for the **guest_all** and one for the $2 \wedge \text{right_virt_checked}$ input
- 11: **if** $i - j = 1$ **then** // hop is our next
- 12: add $\text{SIGN}(\text{TX}_{i,\text{left}}, sk_{\text{loc},\text{virt}}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toRight}}$
- 13: **else if** $j - i = 1$ **then** // hop is our previous
- 14: add $\text{SIGN}(\text{TX}_{i,\text{left}}, sk_{\text{loc},\text{virt}}, \text{ANYPREVOUT})$ 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: $\text{TX}_{\text{next},\text{none}} \leftarrow \text{VIRT.GETEDGETXS}(c_{\text{guest}}, c_P, c_{\bar{P}}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{loc},\text{virt}}, pk_{\text{rem},\text{virt}}, pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}, \text{False})$
- 19: **end if**
- 20: call $\bar{P}.\text{CIRCULATEVIRTUALSIGS}(\text{sigs}_{\text{toRight}})$ and assign returned value to $\text{sigs}_{\text{byRight}}$
- 21: store all signatures in $\text{sigs}_{\text{byRight}}$ that sign any of $\text{TX}_{\text{loc},\text{none}}, \text{TX}_{\text{loc},\text{left}}, \text{TX}_{\text{loc},\text{right}}, \text{TX}_{\text{loc},\text{both}}$ and remove these signatures from $\text{sigs}_{\text{byRight}}$
- 22: ensure that the stored signatures contain one valid signature for $\text{TX}_{\text{loc},\text{right}}$ and $\text{TX}_{\text{loc},\text{both}}$ 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 $2 \wedge \text{right_virt_checked}$ inputs of $\text{TX}_{\text{loc},\text{right}}$ and $\text{TX}_{\text{loc},\text{both}}$ with $pk_{\text{rem},\text{virt}}$ and $pk_{\text{right},\text{guest}}$
- 24: output $(\text{VIRTUALSIGSBACK}, \text{sigs}_{\text{toLeft}}, \text{sigs}_{\text{byRight}})$

Fig. 32.

Process VIRT.INTERMEDIARYSIGS()

```

1: (TXloc,none, TXloc,left, TXloc,right, TXloc,both) ← VIRT.GETMIDTXS(cguest, cP,
   cP̄, csib, csib,rem, pkloc,F, pkrem,F, pksib,F, pksib,rem,F, pkrem,virt, pkloc,virt,
   pkloc,virt, pksib,rem,virt, pkleft,guest, pkright,guest, pkloc,virt, {pksec,i}i ∈ 1...n)
2: // not verifying our signatures in sigsbyLeft, our (trusted) sibling will do that
3: input (VIRTUAL SIGS FORWARD, sigsbyLeft) to sibling
4: VIRT.SIBLINGSIGS()
5: sigstoLeft ← sigsbyRight + sigstoLeft
6: if left_data does not contain data from a second-previous hop then //
   previous hop is host_funder
7:   TXprev,none ← VIRT.GETEDGETXS(cguest, cP̄, cP, pkrem,F, pkloc,F,
   pkrem,virt, pkloc,virt, pkloc,virt, pkleft,guest, pkright,guest, True)
8: end if
9: return sigstoLeft

```

Fig. 33.

Process VIRT.HOSTFUNDEESIGS()

```

1: TXloc,none ← VIRT.GETEDGETXS(cguest, cP, cP̄, pkloc,F, pkrem,F, pkloc,virt,
   pkrem,virt, pkleft,guest, pkright,guest, False)
2: for each hop apart from the first and ours (i ∈ [2, ..., n - 1]) do // hop data
   encoded in left_data
3:   extract data needed for GETMIDTXS() from left_data and assign it to
   datai and {pksec,i}i ∈ 1...n // {pksec,i}i ∈ 1...n contains each party's pki,virt
4:   (TXi,none, TXi,left, TXi,right, TXi,both) ← VIRT.GETMIDTXS(datai,
   {pksec,i}i ∈ 1...n)
5:   add SIGN(TXi,right, skloc,virt, ANYPREVOUT) and SIGN(TXi,both, skloc,virt,
   ANYPREVOUT) to sigstoLeft // 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 ∧ right_virt_checked input
6:   output (SIGN TXS, TXi,left, TXi,right, TXi,both) to fundee and expect reply
   (TXS SIGNED, sigsguest)
7:   add sigsguest to sigstoLeft
8:   if i = n - 1 then // hop is our previous
9:     add SIGN(TXi,left, skloc,virt, ANYPREVOUT) to sigstoLeft
10:  end if
11: end for
12: return sigstoLeft

```

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
   datai and  $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$  //  $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$  contains each party's  $pk_{i,\text{virt}}$ 
3:    $(\text{TX}_{i,\text{none}}, \text{TX}_{i,\text{left}}, \text{TX}_{i,\text{right}}, \text{TX}_{i,\text{both}}) \leftarrow \text{VIRT.GETMIDTXS}(\text{data}_i, \{pk_{\text{sec},i}\}_{i \in 1 \dots n})$ 
4:   add SIGN( $\text{TX}_{i,\text{right}}, sk_{\text{loc},\text{virt}}, \text{ANYPREVOUT}$ ) and SIGN( $\text{TX}_{i,\text{both}}, sk_{\text{loc},\text{virt}}, \text{ANYPREVOUT}$ ) to  $\text{sigs}_{\text{toRight}}$  // if  $i$ -th hop is adjacent, 2 signatures will be
   produced by each SIGN() invocation: one for the guest_all and one for the  $2 \wedge \text{right\_virt\_checked}$  input
5:   output (SIGN TXS,  $\text{TX}_{i,\text{left}}, \text{TX}_{i,\text{right}}, \text{TX}_{i,\text{both}}$ ) to fundee and expect reply
   ( $\text{TXS SIGNED}, \text{sigs}_{\text{guest}}$ )
6:   add  $\text{sigs}_{\text{guest}}$  to  $\text{sigs}_{\text{toRight}}$ 
7:   if  $i = 2$  then // hop is our next
8:     add SIGN( $\text{TX}_{i,\text{left}}, sk_{\text{loc},\text{virt}}, \text{ANYPREVOUT}$ ) to  $\text{sigs}_{\text{toRight}}$ 
9:   end if
10: end for
11: call VIRT.CIRCULATEVIRTUALSIGS( $\text{sigs}_{\text{toRight}}$ ) of  $P$  and assign output to  $\text{sigs}_{\text{byRight}}$ 
12:  $\text{TX}_{\text{loc},\text{none}} \leftarrow \text{VIRT.GETEDGETXS}(c_{\text{guest}}, c_P, c_{\bar{P}}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{loc},\text{virt}}, pk_{\text{rem},\text{virt}}, pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}, \text{True})$ 
13: return (OK)

```

Fig. 35.

Process VIRT.CIRCULATEVIRTUALSIGS($\text{sigs}_{\text{byLeft}}$)

```

1: if  $\text{sigs}_{\text{byLeft}}$  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($\text{sig}_{\text{loc},\text{none}}$)

```

1: if  $\text{sig}_{\text{loc},\text{none}}$  is given as argument then // we are not host_funder
2:   ensure VERIFY( $\text{TX}_{\text{loc},\text{none}}$ ,  $\text{sig}_{\text{loc},\text{none}}$ ,  $pk_{\text{prev},F}$ ) = True //  $pk_{\text{prev},F}$ , found in
   left_data
3:    $\text{sigs}_{\text{loc},\text{none}} \leftarrow \{\text{sig}_{\text{loc},\text{none}}\}$ 
4:   if we have a sibling then // we are not host_fundee
5:     input (VIRTUAL BASE SIG FORWARD,  $\text{sig}_{\text{loc},\text{none}}$ ) to sibling // sibling
     needs  $\text{sig}_{\text{loc},\text{none}}$  for closing
6:      $\text{sigs}_{\text{loc},\text{none}} \leftarrow \{\text{sig}_{\text{loc},\text{none}}\}$ 
7:      $\text{sig}_{\text{next},\text{none}} \leftarrow \text{SIGN}(\text{TX}_{\text{next},\text{none}}, sk_{\text{loc},F})$ 
8:     call VIRT.CIRCULATEVIRTUALSIGS( $\text{sig}_{\text{next},\text{none}}$ ) of  $\bar{P}$  and assign returned
     value to  $\text{sig}_{\text{loc},\text{none}}$ 
9:     ensure VERIFY( $\text{TX}_{\text{loc},\text{none}}$ ,  $\text{sig}_{\text{loc},\text{none}}$ ,  $pk_{\text{next},F}$ ) = True //  $pk_{\text{next},F}$ ,
     found in right_data
10:    add  $\text{sig}_{\text{loc},\text{none}}$  to  $\text{sigs}_{\text{loc},\text{none}}$ 
11:    output (VIRTUAL BASE SIG BACK,  $\text{sig}_{\text{loc},\text{none}}$ ) // sibling needs  $\text{sig}_{\text{loc},\text{none}}$ 
     for closing
12:    add  $\text{sig}_{\text{loc},\text{none}}$  to  $\text{sigs}_{\text{loc},\text{none}}$ 
13:  end if
14:   $\text{sig}_{\text{prev},\text{none}} \leftarrow \text{SIGN}(\text{TX}_{\text{prev},\text{none}}, sk_{\text{loc},F})$ 
15:  return  $\text{sig}_{\text{prev},\text{none}}$ 
16: else // we are host_funder
17:    $\text{sig}_{\text{next},\text{none}} \leftarrow \text{SIGN}(\text{TX}_{\text{next},\text{none}}, sk_{\text{loc},F})$ 
18:   call VIRT.CIRCULATEFUNDINGSIGS( $\text{sig}_{\text{next},\text{none}}$ ) of  $\bar{P}$  and assign returned
     value to  $\text{sig}_{\text{loc},\text{none}}$ 
19:   ensure VERIFY( $\text{TX}_{\text{loc},\text{none}}$ ,  $\text{sig}_{\text{loc},\text{none}}$ ,  $pk_{\text{next},F}$ ) = True //  $pk_{\text{next},F}$  found in
     right_data
20:    $\text{sigs}_{\text{loc},\text{none}} \leftarrow \{\text{sig}_{\text{loc},\text{none}}\}$ 
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  $\leftarrow$  guest.REVOKEPREVIOUS()
5:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
6:   last_poll  $\leftarrow |\Sigma|$ 
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  $\leftarrow$  guest.REVOKEPREVIOUS()
14:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
15:   last_poll  $\leftarrow |\Sigma|$ 
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 (VIRTUAL REVOCATION BACK)
19: end if
20: revoc_for_prev  $\leftarrow$  guest.REVOKEPREVIOUS()
21: output (HOSTS READY) to guest and expect reply (HOST ACK)
22: 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  $R \in \{\text{guest, funder, fundee}\}$ :
2:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
3:    $\text{prev\_went\_on\_chain} \leftarrow \text{TX}_{\text{prev, left}} \in \Sigma \vee \text{TX}_{\text{prev, none}} \in \Sigma$ 
4:    $\text{next\_went\_on\_chain} \leftarrow \text{TX}_{\text{next, right}} \in \Sigma \vee \text{TX}_{\text{next, none}} \in \Sigma$ 
5:    $\text{last\_poll} \leftarrow |\Sigma|$ 
6:   if  $\text{prev\_went\_on\_chain} \vee \text{next\_went\_on\_chain}$  then
7:     ignore all messages except for (CHECK CHAIN FOR CLOSING) by  $R$ 
8:      $\text{State} \leftarrow \text{CLOSING}$ 
9:   end if
10:  if  $\text{prev\_went\_on\_chain} \wedge \text{next\_went\_on\_chain}$  then
11:    VIRT.SIGNANDSUBMIT( $\text{TX}_{\text{loc, both}}, \text{sigs}_{\text{loc, both}}$ )
12:  else if  $\text{prev\_went\_on\_chain}$  then
13:    VIRT.SIGNANDSUBMIT( $\text{TX}_{\text{loc, left}}, \text{sigs}_{\text{loc, left}}$ )
14:  else if  $\text{next\_went\_on\_chain}$  then
15:    VIRT.SIGNANDSUBMIT( $\text{TX}_{\text{loc, right}}, \text{sigs}_{\text{loc, right}}$ )
16:  end if

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

```

Fig. 39.

Process VIRT – close

```

1: On input (CLOSE) by  $R \in \{\text{guest}, \text{funder}, \text{fundee}\}$ : // At most one of funder,
   fundee is defined
2:   if  $State = \text{CLOSED}$  then output (CLOSED) to  $R$ 
3:   if  $State = \text{GUEST PUNISHED}$  then output (GUEST PUNISHED) to  $R$ 
4:   ensure  $State = \text{OPEN}$ 
5:   if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$  then //  $\text{host}_P$  is a VIRT
6:     ignore all messages except for output (CLOSED) by  $\text{host}_P$ . Also relay to
      $\text{host}_P$  any (CHECK CHAIN FOR CLOSING) input received
7:     input (CLOSE) to  $\text{host}_P$ 
8:   end if
9:   // if we have a  $\text{host}_P$ , 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  $\text{tx}$  be the unique valid TX for  $\Sigma$  among  $(\text{TX}_{\text{loc}, \text{none}}, \text{TX}_{\text{loc}, \text{left}},$ 
    $\text{TX}_{\text{loc}, \text{right}}, \text{TX}_{\text{loc}, \text{both}})$  // if we are not an intermediary, only the first exists
12:  let  $\text{sigs}$  be the corresponding set of signatures among  $(\text{sigs}_{\text{loc}, \text{none}},$ 
    $\text{sigs}_{\text{loc}, \text{left}}, \text{sigs}_{\text{loc}, \text{right}}, \text{sigs}_{\text{loc}, \text{both}})$ 
13:  add  $\text{SIGN}(\text{tx}, sk_{A,F})$  and  $\text{SIGN}(\text{tx}, sk_{\text{loc}, \text{virt}})$  to  $\text{sigs}$  // 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:   $State \leftarrow \text{CLOSING}$ 
16:  send (SUBMIT,  $(\text{tx}, \text{sigs})$ ) to  $\mathcal{G}_{\text{Ledger}}$ 

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

29: On input (USED REVOCATION) by guest: // (USED REVOCATION) by
   funder/fundee is ignored
30:   $State \leftarrow \text{GUEST PUNISHED}$ 
31:  input (USED REVOCATION) to  $\text{host}_P$ , expect reply (USED REVOCATION OK)
32:  if funder or fundee is defined then output (ENABLER USED REVOCATION)
   to it

33: On output (USED REVOCATION) by  $\text{host}_P$ :
34:   $State \leftarrow \text{GUEST PUNISHED}$ 
35:  if funder or fundee is defined then output (ENABLER USED REVOCATION)
   to it

```

Fig. 40.

Lemma 1 (Real world balance security). Consider a real world execution with $P \in \{\text{Alice}, \text{Bob}\}$ 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 $(\text{FUND ME}, f_i, \dots)$ 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 (PAY, e_i, \dots) by \bar{P} while State was OPEN and P subsequently transitioned to OPEN State] l times.

Let $\phi = 1$ if $P = \text{Alice}$, or $\phi = 0$ if $P = \text{Bob}$. If P receives (CLOSE) by \mathcal{E} and, if $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$ 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

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

with overwhelming probability in the security parameter.

Proof. P can transition to the OPEN State for the first time only if it:

- has received (OPEN, c, \dots) while in the INIT State by either \mathcal{E} if $(P = \text{Alice}, \text{Fig. 15, l. 1})$ or \bar{P} (if $P = \text{Bob}, \text{Fig. 3, l. 3})$,
- it internally holds a signature on the commitment transaction $C_{P,0}$ that is valid by $pk_{\bar{P},F}$ (Fig. 12, ll. 12 and 23),
- it either has the transaction F in the $\mathcal{G}_{\text{Ledger}}$ state if $\text{host} = \mathcal{G}_{\text{Ledger}}$ (Fig. 13, l. 3), or is able to cause an eventual inclusion of a $(c, 2/\{pk_{A,F}, pk_{B,F}\})$ output in the $\mathcal{G}_{\text{Ledger}}$ state by passing input (CLOSE) to host . The latter is true if $\text{hops}[0].\text{left}$ (when $P = \text{Alice}, \text{Fig. 11, l. 6})$ or $\text{hops}[-1].\text{right}$ (when $P = \text{Bob}, \text{Fig. 14, l. 8})$ has passed output (FUNDED, \dots) to P , as the outputting party is trusted (as it executes locally) and has itself been passed output (HOSTS READY) by its (also trusted) host VIRT ITI (Fig. 17, l. 12). The host only outputs (HOSTS READY) when its guest (which is the aforementioned party that output (FUNDED, \dots)) verifies that the remote revocation has happened correctly (Fig. 38, ll. 2 or 8). As we will see later, this verification ensures that either the aforementioned output will end up in the $\mathcal{G}_{\text{Ledger}}$ state, or the guest ’s counterparty will lose all their funds to the guest .

□

Lemma 2 (Ideal world balance). Consider an ideal world execution with functionality $\mathcal{F}_{\text{Chan}}$ and simulator \mathcal{S} . Let $P \in \{\text{Alice}, \text{Bob}\}$ one of the two parties of $\mathcal{F}_{\text{Chan}}$. Assume that all of the following are true:

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

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

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

Proof. We will prove the Lemma by following the evolution of the balance_P variable.

- When $\mathcal{F}_{\text{Chan}}$ is activated for the first time, it sets $\text{balance}_P \leftarrow 0$ (Fig. 2, l. 1).
- If $P = \text{Alice}$ and it receives (OPEN, c, \dots) by \mathcal{E} , it stores c (Fig. 2, l. 10). If later $State_P$ becomes OPEN, $\mathcal{F}_{\text{Chan}}$ sets $\text{balance}_P \leftarrow c$ (Fig. 2, ll. 13 or 29). In contrast, if $P = \text{Bob}$, it is $\text{balance}_P = 0$ 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 $State_P = \text{OPEN}$, P stores f_i (Fig. 4, l. 1). The next time $State_P$ transitions to OPEN (if such a transition happens), balance_P is decremented by f_i (Fig. 4, l. 22). Therefore, if this cycle happens $n \geq 0$ times, 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 $State_P = \text{OPEN}$, d_i is stored (Fig. 3, l. 2). The next time $State_P$ transitions to OPEN (if such a transition happens), balance_P is decremented by d_i (Fig. 3, l. 13). Therefore, if this cycle happens $m \geq 0$ times, balance_P will be decremented by $\sum_{i=1}^m d_i$ in total.
- Every time P receives input $(\text{GET PAID}, e_i)$ by \mathcal{E} while $State_P = \text{OPEN}$, e_i is stored (Fig. 3, l. 7). The next time $State_P$ transitions to OPEN (if such a transition happens) balance_P is incremented by e_i (Fig. 3, l. 19). Therefore, if this cycle happens $l \geq 0$ times, 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 $(\text{CLOSE},$

$P)$ by \mathcal{S} , it is $\text{balance}_P = c - \sum_{i=1}^n f_i - \sum_{i=1}^m d_i + \sum_{i=1}^l e_i$ if $P = \text{Alice}$, or else if $P = \text{Bob}$, $\text{balance}_P = - \sum_{i=1}^n f_i - \sum_{i=1}^m d_i + \sum_{i=1}^l e_i$. \square

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 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 **negligent** (Fig. 7, l. 4), which means that the first condition of Lemma 1 is violated. In the second case, it is $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$ and the state of host_P is GUEST PUNISHED (Fig. 40, ll. 29 or 33), so in case P receives (CLOSE) by \mathcal{E} the output of 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 $P = \text{Alice}$, it must receive (INIT, pk) by \mathcal{E} when $\text{State}_P = \text{UNINIT}$, then either receive (OPEN, c , $\mathcal{G}_{\text{Ledger}}$, ...) by \mathcal{E} and (BASE OPEN) by \mathcal{S} or (OPEN, c , **hops** ($\neq \mathcal{G}_{\text{Ledger}}$), ...) by \mathcal{E} , (FUNDED, HOST, ...) 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 $P = \text{Alice}$, then the second condition of Lemma 1 holds as well. The same line of reasoning can be used to deduce that if $P = \text{Bob}$, 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 , ...) 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 $\text{State}_{\bar{P}} = \text{IGNORED}$, then State_P transitions directly back to OPEN. If on the other hand $\text{State}_{\bar{P}} = \text{OPEN}$ and $\mathcal{F}_{\text{Chan}}$ intercepts a similar VIRT ITI definition command through \bar{P} , $\text{State}_{\bar{P}}$ transitions to TENTATIVE HELP FUND. On receiving the aforementioned (FUND) message by \mathcal{S} and given that $\text{State}_{\bar{P}} = \text{TENTATIVE HELP FUND}$, $\mathcal{F}_{\text{Chan}}$ also sets $\text{State}_{\bar{P}}$ to SYNC HELP FUND. Then both $\text{State}_{\bar{P}}$ 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 , ...) 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} , $State_P$ transitions 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 $State_{\bar{P}} = \text{IGNORED}$, then $State_P$ transitions directly back to OPEN. If on the other hand $State_{\bar{P}} = \text{OPEN}$ 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 $State_{\bar{P}} = \text{TENTATIVE GET PAID}$, $\mathcal{F}_{\text{Chan}}$ 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 transitioned to the OPEN state (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 $State_P = \text{IGNORED}$ 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 $State_P \neq \text{IGNORED}$ (Fig. 5, l. 3). We deduce that, once $State_P = \text{IGNORED}$, 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 $P = Alice$, 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. \square

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,*

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

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