

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

**Fig. 1.**

**Functionality  $\mathcal{F}_{\text{Chan}}$  – state machine, Pt. 1**

```

1: On initialisation:
2:    $pk_A \leftarrow \perp$ ;  $pk_B \leftarrow \perp$ 
3:    $\text{balance}_A \leftarrow 0$ ;  $\text{balance}_B \leftarrow 0$ 
4:    $\text{is\_corrupted\_or\_negligent}_A \leftarrow \text{False}$ ;  $\text{is\_corrupted\_or\_negligent}_B \leftarrow$ 
   False;
5:    $\text{State} \leftarrow \perp$ 

6: On (BECAME CORRUPTED OR NEGLIGENT,  $P$ ) by  $\mathcal{A}$ :
7:    $\text{is\_corrupted\_or\_negligent}_P \leftarrow \text{True}$ 

8: On (INIT,  $pk$ ) to  $P$  by  $\mathcal{E}$ :
9:   if  $P = \text{Bob}$  then  $\bar{P} \leftarrow \text{Alice}$  else  $\bar{P} \leftarrow \text{Bob}$ 
10:  ensure  $\text{State} \in \{\perp, \text{INIT}_P\}$ 
11:   $pk_P \leftarrow pk$ 
12:  if  $\text{State} = \perp$  then  $\text{State} \leftarrow \text{TENTATIVE-INIT}_P$  else  $\text{State} \leftarrow \text{TENTATIVE-INIT}$ 

13: On (INIT,  $P$ ) by  $\mathcal{A}$ :
14:  if  $\text{State} = \text{TENTATIVE-INIT}_P$  then  $\text{State} \leftarrow \text{INIT}_P$ 
15:  if  $\text{State} = \text{TENTATIVE-INIT}$  then  $\text{State} \leftarrow \text{INIT}$ 

16: On (OPEN,  $x, \dots$ ) to Alice by  $\mathcal{E}$ :
17:  ensure  $\text{State} = \text{INIT}$ 
18:   $\text{balance}_A \leftarrow x$ ;  $\text{balance}_B \leftarrow 0$ 
19:   $\text{State} \leftarrow \text{TENTATIVE-OPEN}$ 

20: On (OPEN) by  $\mathcal{A}$ :
21:  ensure  $\text{State} = \text{TENTATIVE-OPEN}$ 
22:   $\text{State} \leftarrow \text{OPEN}$ 

23: On (PAY,  $x$ ) to  $P$  by  $\mathcal{E}$ :
24:  ensure  $\text{State} = \text{OPEN}$ 
25:   $c \leftarrow x$ 
26:  if  $P = \text{Alice}$  then  $S \leftarrow \text{Alice}$ ;  $\bar{S} \leftarrow \text{Bob}$  else  $S \leftarrow \text{Bob}$ ;  $\bar{S} \leftarrow \text{Alice}$ 
27:   $\text{State} \leftarrow \text{PAYING}$ 

28: On (PAY OK) by  $\mathcal{A}$ :
29:  ensure  $\text{State} = \text{PAYING}$ 
30:   $\text{balance}_S \leftarrow \text{balance}_S - c$ ;  $\text{balance}_{\bar{S}} \leftarrow \text{balance}_{\bar{S}} + c$ 
31:  forget  $c, S, \bar{S}$ 
32:   $\text{State} \leftarrow \text{OPEN}$ 

```

**Fig. 2.**

**Functionality  $\mathcal{F}_{\text{Chan}}$  – state machine, Pt. 2**

```

1: On (FUND ME,  $x, \dots$ ) to  $P$  by  $R$ :
2:   ensure  $R$  is an ITI that runs  $\mathcal{F}_{\text{Chan}}$  or  $\Pi_{\text{Chan}}$  code
3:   ensure  $State = \text{OPEN}$ 
4:    $c \leftarrow x$ 
5:   if  $P = \text{Alice}$  then  $S \leftarrow \text{Alice}$  else  $S \leftarrow \text{Bob}$ 
6:    $State \leftarrow \text{FUNDING}$ 

7: On (FUND OK) by  $\mathcal{A}$ :
8:   ensure  $State = \text{FUNDING}$ 
9:    $balance_S \leftarrow balance_S - c$ 
10:  forget  $c, S$ 
11:   $State \leftarrow \text{OPEN}$ 

12: On (CLOSE) by  $\mathcal{E}$ :
13:  ensure  $State = \text{OPEN}$ 
14:   $State \leftarrow \text{CLOSING}$ 

15: On (CLOSE) by  $\mathcal{A}$ :
16:  ensure  $State \in \{\text{OPEN}, \text{CLOSING}, \text{PAYING}, \text{FUNDING}\}$ 
17:  for  $P \in \{A, B\}$  do
18:    if  $is\_corrupted\_or\_negligent_P = \text{False}$  then
19:      input (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $P$  and assign output to  $\Sigma_P$ 
20:       $coins_P \leftarrow$  sum of coins exclusively spendable by  $pk_P$  in  $\Sigma_P$ 
21:    end if
22:  end for
23:   $check_A \leftarrow is\_corrupted\_or\_negligent_A \vee coins_A \geq balance_A$ 
24:   $check_B \leftarrow is\_corrupted\_or\_negligent_B \vee coins_B \geq balance_B$ 
25:  if  $check_A \wedge check_B$  then
26:     $State \leftarrow \text{CLOSED}$ 
27:  else // balance security is broken
28:    halt
29:  end if

```

**Fig. 3.**

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

- On receiving  $(\text{RELAY}, \text{in\_msg}, P, R, \text{in\_mode})$  by  $\mathcal{F}_{\text{Chan}}$  ( $\text{in\_mode} \in \{\text{input}, \text{output}, \text{network}\}$ ,  $P \in \{\text{Alice}, \text{Bob}\}$ ), handle  $(\text{in\_msg})$  with the simulated party  $P$  as if it was received from  $R$  by means of  $\text{in\_mode}$ . In case simulated  $P$  does not exist yet, initialise it as an LN ITI. If there is a resulting message  $\text{out\_msg}$  that is to be sent by simulated  $P$  to  $R'$  by means of  $\text{out\_mode} \in \{\text{input}, \text{output}, \text{network}\}$ , send  $(\text{RELAY}, \text{out\_msg}, P, R', \text{out\_mode})$  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  $(\text{msg})$  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. 4.**

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

- When referring to a player  $P$ , it must be one of the parties that correspond to  $\mathcal{F}_{\text{Chan}}$ .
  - When an action in the current Figure interrupts the simulation of a party, the latter is continued from the interruption location after control is handed back by  $\mathcal{F}_{\text{Chan}}$ .
- 1: On (CORRUPT) by  $\mathcal{A}$ , addressed to  $P$ :
  - 2:     // After executing this code, deliver (CORRUPT) to simulated  $P$  as detailed in Fig. 4. Given that  $\mathcal{F}_{\text{Chan}}$  returns control directly to us after it handles this message, we will always deliver (CORRUPT) successfully.
  - 3:     send (INFO, BECAME CORRUPTED OR NEGLIGENT,  $P$ ) to  $\mathcal{F}_{\text{Chan}}$
  - 4: When simulated  $P$  sets its internal variable **negligent** to True (Fig. 7, l. 26/Fig. 6, l. 7):
  - 5:     send (INFO, BECAME CORRUPTED OR NEGLIGENT,  $P$ ) to  $\mathcal{F}_{\text{Chan}}$
  - 6: When simulated  $P$  moves to the INIT state (Fig. 6, l. 21):
  - 7:     send (INFO, INIT,  $P$ ) to  $\mathcal{F}_{\text{Chan}}$
  - 8: When the last of the honest simulated  $\mathcal{F}_{\text{Chan}}$ 's parties moves to the OPEN state for the first time (Fig. 10, l. 19/Fig. 12, l. 5/Fig. 13, l. 18):
  - 9:     send (INFO, OPEN) to  $\mathcal{F}_{\text{Chan}}$
  - 10: When (both  $\mathcal{F}_{\text{Chan}}$ 's simulated parties are honest and one party completes sending a payment (Fig. 18, l. 6) and the counterparty completes receiving that payment (Fig. 18, l. 19)), or (when only one of the two is honest and (completes either receiving or sending a payment)): // also send this message if both parties are honest when Fig. 18, l. 6 is executed by one party, but its counterparty is corrupted before executing Fig. 18, l. 19
  - 11:     send (INFO, PAY OK) to  $\mathcal{F}_{\text{Chan}}$
  - 12: When (both  $\mathcal{F}_{\text{Chan}}$ 's simulated parties are honest and they both complete changing to a new host (Fig. 7, l. 12)), or (when only one of the two is honest and it completes changing to a new host): // also send this message if both parties are honest when the change of host is executed by one party, but its counterparty is corrupted before changing host
  - 13:     send (INFO, FUND OK) to  $\mathcal{F}_{\text{Chan}}$
  - 14: When one of the honest simulated  $\mathcal{F}_{\text{Chan}}$ 's parties moves to the CLOSED state (Fig. 21, l. 8/Fig. 23, l. 26):
  - 15:     send (INFO, CLOSE) to  $\mathcal{F}_{\text{Chan}}$

**Fig. 5.**

**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. 6.**

**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. 7.**

**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. 8.

**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:    $host_A \leftarrow \mathcal{G}_{\text{Ledger}}$ 
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. 9.



**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, host_fundee) by hops[-1].right (c.f. Fig 12, line 10)
8: if 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 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. 10.**

**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. 11.**

**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. 12.**

**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. 13.**

**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  $\text{VERIFY}(C_{\bar{P},i+1}, \text{sig}_{P,C,i+1}, pk'_{P,F}) = \text{True}$ 
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  $\text{VERIFY}(C_{P,i+1}, \text{sig}_{\bar{P},C,i+1}, pk'_{\bar{P},F}) = \text{True}$ 

```

**Fig. 14.**

**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  $c_P - \text{locked}_P \geq c$ 
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. 7, 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:     output (FUNDED,  $\text{host}_P$ , layer) to funder // do not expect reply by
   funder
21:   end if
22:   reply (HOST ACK)

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

```

**Fig. 15.**

**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. 16.**

**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 = OPEN
- 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: **reply** (COMMITMENT SIGNED,  $\text{sig}_{\bar{P},C,i+1}$ ,  $\text{sig}_{\bar{P},R,i}$ )
- 16:  $C_{P,i+1} \leftarrow C_{P,i}$  with  $x$  coins moved from  $P$ 's to  $\bar{P}$ 's output

**Fig. 17.**

**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 (EW BALANCE,  $c_P$ ,  $c_{\bar{P}}$ ) to host_P
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:  $R_{\bar{P},i} \leftarrow \text{TX}$  {input:  $C_{P,i}.\text{outputs}.\bar{P}$ , output: ( $c_P$ ,  $pk_{\bar{P},\text{out}}$ )}
18: ensure VERIFY( $R_{\bar{P},i}$ ,  $\text{sig}_{P,R,i}$ ,  $pk_{P,R}$ ) = True
19: add  $x$  to paid_in
20:  $c_P \leftarrow c_P - x$ ;  $c_{\bar{P}} \leftarrow c_{\bar{P}} + x$ ;  $i \leftarrow i + 1$ 
21: if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$  has a host_sibling then // we are intermediary
   channel
22:  input (NEW BALANCE,  $c_{\bar{P}}$ ,  $c_P$ ) to host_P
23:  relay message as input to sibling // run by VIRT
24:  relay message as output to guest // run by VIRT
25:  store new sibling balance and reply (NEW BALANCE OK)
26:  output (NEW BALANCE OK) to sibling // run by VIRT
27:  output (NEW BALANCE OK) to guest // run by VIRT
28: end if

```

**Fig. 18.**

**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. 19.**

**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  $\text{host}_P$ 
3: end if

```

**Fig. 20.**

**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. 21.**



**Process LN.SUBMITANDCHECKREVOCATION( $j$ )**

```

1:  $\text{sig}_{P,R,j} \leftarrow \text{SIGN}(R_{P,j}, \text{sk}_{P,R})$ 
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

```

**Fig. 22.**

**Process LN – On (CLOSE):**

```

1: ensure  $\text{State} \notin \{\perp, \text{INIT}, \text{TOPPED UP}\}$  // channel open
2: if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$  then // we have a virtual channel
3:    $\text{State} \leftarrow \text{HOST CLOSING}$ 
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: end if
6:  $\text{State} \leftarrow \text{CLOSING}$ 
7: input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
8: if  $C_{\bar{P},i} \in \Sigma$  then // counterparty has closed honestly
9:   no-op // do nothing
10: else if  $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$  then // counterparty has closed maliciously
11:   LN.SUBMITANDCHECKREVOCATION( $j$ )
12: else // counterparty is idle
13:   while  $\nexists$  unspent output  $\in \Sigma$  that  $C_{P,i}$  can spend do // possibly due to an
     active timelock
14:     wait for input (CHECK VIRTUAL) // ignore other messages
15:     input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
16:   end while
17:   // provably reachable – c.f. TODO: ref
18:    $\text{sig}'_{P,C,i} \leftarrow \text{SIGN}(C_{P,i}, \text{sk}_{P,F})$ 
19:   input (SUBMIT,  $(C_{P,i}, \text{sig}_{P,C,i}, \text{sig}'_{P,C,i})$ ) to  $\mathcal{G}_{\text{Ledger}}$ 
20:   while  $C_{P,i} \notin \Sigma$  do
21:     wait for input (CHECK CLOSED) // ignore other messages
22:     input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
23:   end while
24:   // provably reachable – c.f. TODO: ref
25: end if
26:  $\text{State} \leftarrow \text{CLOSED}$ 
27: output (CLOSED)

```

**Fig. 23.**

**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, hostP,  $\bar{P}$ , 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,  $\bar{P}$ , hostP,  $c_{\text{guest}}$ ,  $pk_{\text{left}, \text{guest}}$ ,  $pk_{\text{right}, \text{guest}}$ ,
    ( $sk_{\text{loc}, \text{virt}}$ ,  $pk_{\text{loc}, \text{virt}}$ ), ( $sk_{\text{loc}, F}$ ,  $pk_{\text{loc}, F}$ ),  $pk_{\text{rem}, F}$ ,  $pk_{\text{rem}, \text{virt}}$ ) by guest:
16:   last_poll  $\leftarrow \perp$ 
17:   ensure 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. 24.**

**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. 25.**

**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. 26.**

**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. 27.**

**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  $\text{data}_i$  and  $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$  //  $P$  and  $\text{comm\_keys}$  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. 28.**

**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. 29.

**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. 30.

**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. 31.**

**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. 32.**



**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. 33.**

**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. 34.**

**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:    $\text{prev\_went\_on\_chain} \leftarrow \text{TX}_{\text{prev, left}} \in \Sigma \wedge \text{TX}_{\text{prev, both}} \in \Sigma$ 
4:    $\text{next\_went\_on\_chain} \leftarrow \text{TX}_{\text{next, right}} \in \Sigma \wedge \text{TX}_{\text{next, both}} \in \Sigma$ 
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. 35.**

**Process VIRT – close**

```

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

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

```

**Fig. 36.**

**Lemma 1 (Real world balance security).** *Let  $P$  honest LN ITI such that all of the following are true:*

- the internal variable *negligent* of  $P$  has value “False”,
- $P$  has transitioned to the OPEN State for the first time after having received  $(\text{OPEN}, c, \dots)$  by either  $\mathcal{E}$  or another LN ITI  $\bar{P}$ ,

- $P$  [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 (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  $\mathcal{E}$ , 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

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

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

- the internal variable `is_corrupted_or_negligentP` of  $\mathcal{F}_{\text{Chan}}$  has the value “False”,
- If  $P = \text{Alice}$ ,  $P$  has received (OPEN,  $c, \dots$ ) by  $\mathcal{E}$ ,
- $P$  [has received (FUND ME,  $f_i, \dots$ ) as input by another  $\mathcal{F}_{\text{Chan}}$  or  $\Pi_{\text{Chan}}$  ITI while State was OPEN and subsequently  $\mathcal{F}_{\text{Chan}}$  transitioned to OPEN State]  $n$  times,
- $P$  [has received (PAY,  $d_i$ ) by  $\mathcal{E}$  while State was OPEN and  $\mathcal{F}_{\text{Chan}}$  subsequently transitioned to OPEN State]  $m$  times,
- $\bar{P}$  [has received (PAY,  $e_i$ ) by  $\mathcal{E}$  while State was OPEN and  $\mathcal{F}_{\text{Chan}}$  subsequently transitioned to OPEN State]  $l$  times.

Let  $\phi = 1$  if  $P = \text{Alice}$ , or  $\phi = 0$  if  $P = \text{Bob}$ . If  $P$  or  $\bar{P}$  receives (CLOSE) by  $\mathcal{S}$ , then

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

**Lemma 3 (No halt).** In an ideal execution with  $\mathcal{F}_{\text{Chan}}$  and  $\mathcal{S}$ , the functionality never halts (i.e. l. 28 of Fig. 3 is never executed).

*Proof.* The only way for  $\mathcal{F}_{\text{Chan}}$  to halt is if either `checkA` or `checkB` fails. For these checks to be run,  $\mathcal{F}_{\text{Chan}}$  must have received (CLOSE) by  $\mathcal{S}$  and must have been in the OPEN State (as any other state of Fig. 3, l. 16 can only be reached after  $\mathcal{F}_{\text{Chan}}$  has been in the OPEN State). Additionally,  $\mathcal{F}_{\text{Chan}}$  can only reach the OPEN State if all honest simulated parties transition to the OPEN State as well (Fig. 5, l. 9), which in turn happens for simulated *Alice* only if she has received (OPEN,  $\dots$ ) by  $\mathcal{E}$ . Observe further that  $\mathcal{S}$  notifies  $\mathcal{F}_{\text{Chan}}$  right away when either simulated party becomes corrupted or negligent (Fig. 5, ll. 3 and 5 respectively) and the balance of each party is checked by  $\mathcal{F}_{\text{Chan}}$  only if this party is not corrupted nor negligent. These facts in combination mean that for each party,

whenever the prerequisites for Lemma 2 are true the prerequisites for Lemma 1 are also true and therefore the check for this party will succeed (c.f. Fig 3, l. 20). Furthermore, when the prerequisites for Lemma 2 do not hold, the check for the respective party will be true. On aggregate, when an ideal execution of  $\mathcal{F}_{\text{Chan}}$  and  $\mathcal{S}$  take place, in no case will  $\mathcal{F}_{\text{Chan}}$  halt.  $\square$

**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,*

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

*Proof.* By inspection of Figs. 1 and 4 we can deduce that for a particular  $\mathcal{E}$ , in the ideal world execution  $\text{EXEC}_{\mathcal{S}, \mathcal{A}, \mathcal{E}}^{\mathcal{F}_{\text{Chan}}, \mathcal{G}_{\text{Ledger}}}$ ,  $\mathcal{S}_{\mathcal{A}}$  simulates internally the two LN parties exactly as they would execute in the real world execution,  $\text{EXEC}_{\text{LN}, \mathcal{A}, \mathcal{E}}^{\mathcal{G}_{\text{Ledger}}}$  in case  $\mathcal{F}_{\text{Chan}}$  never halts. The latter is true according to Lemma 3, therefore the two executions are information-theoretically indistinguishable.  $\square$

**Theorem 2.** *Assume that at the end of the execution,  $\mathcal{G}_{\text{Ledger}}$  contains exactly one “groups” transaction that precedes all “funding” transactions and contains as payload a partition  $\mathcal{G}$  into groups of all VChan parties, with each group containing the parties that belong to the same (human) owner. Then the following holds:*

$$\begin{aligned} & \forall G \in \mathcal{G} \text{ such that all parties in } G \text{ are honest,} \\ & \sum_{P \in G} \text{logged-coins}(P) = \sum_{P \in G} \text{ledger-coins}(P) = \\ & = \sum_{P \in G} (\text{top-up}(P) + \sum_{m \in \mathcal{T}} \text{pay-in}(m, P) - \sum_{m \in \mathcal{T}} \text{pay-out}(m, P)) , \end{aligned}$$

where  $\mathcal{T}$  is the execution transcript and:

$\text{logged-coins}(P) = c_P$ , as recorded in  $\mathcal{F}_{\text{Chan}}/\Pi_{\text{Chan}}$

$\text{ledger-coins}(P) = \text{coins spendable with the secret key } sk \text{ of } P \text{ if the closing transactions of all open channels are submitted to } \mathcal{G}_{\text{Ledger}} \text{ and added to the state of all parties and then } t \text{ new blocks enter the state of all honest parties}$

$$\begin{aligned} \text{top-up}(P) &= \begin{cases} c_{\text{on}}, & \text{as determined on message (CHECK TOP UP),} \\ & \text{if such a message was handled} \\ 0, & \text{otherwise} \end{cases} \\ \text{pay-in}(m, P) &= \begin{cases} x, & \text{if message } m \text{ updated the channel to} \\ & \text{a state in which } P \text{ had } x \text{ more coins} \textcolor{red}{\text{TODO: improve prev}} \\ 0, & \text{otherwise} \end{cases} \\ \text{pay-out}(m, P) &= \begin{cases} x, & \text{if } m = (\text{PAY}, x) \text{ was received by } P \text{ and} \\ & P \text{ output (PAY SUCCESS) as a result} \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

## References