

## 1 Future work

- Add support for cooperative adding multiple virtuals to single channel (needs cooperation by all hops of all existing virtuals of current channel)
- Add support for cooperative closing
- Use eltoo instead of lightning to avoid balance restriction that prevents the revoked-griefing attack
- Allow for user-defined “leeway” timeout and timeout renegotiation
- Incorporate fees
- Prevent DoS attacks

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

- On receiving  $(\text{msg})$  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  $\text{msg}$  to  $R$  as  $P$  by means of  $\text{mode}$ . //  $\mathcal{A}$  fully controls outgoing messages by  $\mathcal{F}_{\text{Chan}}$
- On receiving  $(\text{INFO}, \text{msg})$  by  $\mathcal{A}$ , handle  $(\text{msg})$  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  $\text{host}_P$  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,  $\text{host}$ ,  $\dots$ ) to *Alice* by  $\text{enabler}_A$  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,  $\text{host}$ ,  $\dots$ ) 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  $\Sigma$
- 5:      $\text{coins}_P \leftarrow$  sum of coins exclusively spendable by  $pk_P$  in  $\Sigma$
- 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,  $\text{in\_msg}$ ,  $P$ ,  $R$ ,  $\text{in\_mode}$ ) 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 (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. 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 21 respectively), or (when only one party is honest and (completes either receiving or sending a payment)): // also send this message if both parties are honest when Fig. 20, l. 6 is executed by one party, but its counterparty is corrupted before executing Fig. 20, l. 21
  - 17:     send (INFO, PAY) to  $\mathcal{F}_{\text{Chan}}$
  - 18: When honest  $P$  executes Fig. 17, l. 20 or (when honest  $P$  executes Fig. 17, l. 18 and  $\bar{P}$  is corrupted): // in the first case if  $\bar{P}$  is honest, it has already moved to the new host, (Fig 38, ll. 7, 23): lifting to next layer is done
  - 19:     send (INFO, FUND) to  $\mathcal{F}_{\text{Chan}}$
  - 20: When one of the honest simulated  $\mathcal{F}_{\text{Chan}}$ ’s parties  $P$  moves to the CLOSED state (Fig. 24, l. 8 or l. 11):
  - 21:     send (INFO, CLOSE,  $P$ ) to  $\mathcal{F}_{\text{Chan}}$

**Fig. 7.**

**Process LN – init**

```

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  $\text{last\_poll} + p < |\Sigma|$  then //  $p$  is a global parameter
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:    $R_{\bar{P},i} \leftarrow \text{TX } \{\text{input: } C_{P,i}.\text{outputs}.P, \text{ output: } (C_{P,i}.\text{outputs}.P.\text{value},$ 
       $pk_{\bar{P},\text{out}})\}$ 
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:    $R_{P,i} \leftarrow \text{TX } \{\text{input: } C_{\bar{P},i}.\text{outputs}.P, \text{ output: } (C_{\bar{P},i}.\text{outputs}.\bar{P}.\text{value},$ 
       $pk_{P,\text{out}})\}$ 
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:    $t_P \leftarrow s + p$  //  $s$  is the upper bound of  $\eta$  from Lemma 7.19 of [1]
12:    $State \leftarrow \text{WAITING FOR COMM SIG}$ 
13: else // opening virtual channel
14:    $State \leftarrow \text{WAITING FOR CHECK KEYS}$ 
15: end if
16: reply (ACCEPT CHANNEL,  $pk_{B,F}$ ,  $pk_{B,R}$ ,  $pk_{B,out}$ )
17: ensure  $State = \text{WAITING FOR OPENING KEYS}$ 
18: store  $pk_{B,F}$ ,  $pk_{B,R}$ ,  $pk_{B,out}$ 
19:  $State \leftarrow \text{OPENING KEYS OK}$ 

```

Fig. 10.

**Process LN.PREPAREBASE()**

```

1: if hops =  $\mathcal{G}_{\text{Ledger}}$  then // opening base channel
2:    $F \leftarrow \text{TX}$  {input:  $(c, pk_{A,chain})$ , output:  $(c, 2/\{pk_{A,F}, pk_{B,F}\})$ }
3:    $host_P \leftarrow \mathcal{G}_{\text{Ledger}}$ 
4:   layer  $\leftarrow 0$ 
5:    $t_P \leftarrow s + p$ 
6: else // opening virtual channel
7:   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,  $t_P$ ) // ignore any other
   message
8:   layer  $\leftarrow$  funder_layer
9: 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:  $(c, 2/\{pk_{A,F}, pk_{B,F}\})$ , 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:  $(c, 2/\{pk_{A,F}, pk_{B,F}\})$ , 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 (Fig 14, l. 10)
8: if hops =  $\mathcal{G}_{\text{Ledger}}$  then // opening base channel
9:    $F \leftarrow \text{TX}$  {input:  $(c, pk_{A,\text{chain}})$ , output:  $(c, 2/\{pk_{A,F}, pk_{B,F}\})\}$ 
10: end if
11:  $C_{B,0} \leftarrow \text{TX}$  {input:  $(c, 2/\{pk_{A,F}, pk_{B,F}\})$ , outputs:  $(c, pk_{A,\text{out}})$ ,  $(0,$ 
    $(pk_{B,\text{out}} + (t + s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})\}$ 
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:  $(c, 2/\{pk_{A,F}, pk_{B,F}\})$ , 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. 12.**

**Process LN.COMMITBASE()**

```

1:  $\text{sig}_F \leftarrow \text{SIGN}(F, sk_{A,\text{chain}})$ 
2: input (SUBMIT,  $(F, \text{sig}_F)$ ) to  $\mathcal{G}_{\text{Ledger}}$  // enter “while” below before sending
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,  $host_P$ ,  $funder\_layer$ ,  $t_P$ ) 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,
   reducing the input and  $\bar{P}$ 's output by  $c_{\text{virt}}$ 
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,
   reducing the input and  $\bar{P}$ 's output by  $c_{\text{virt}}$ 
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,
   reducing the input and  $P$ 's output by  $c_{\text{virt}}$ 
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,
   reducing the input and  $P$ 's output by  $c_{\text{virt}}$ 
11: ensure  $\text{VERIFY}(C_{P,i+1}, \text{sig}_{\bar{P},C,i+1}, pk'_{\bar{P},F}) = \text{True}$ 

```

**Fig. 16.**

**Process LN – virtualise start and end**

```

1: On input (FUND ME,  $c_{\text{virt}}$ , fundee, hops,  $pk_{A,V}$ ,  $pk_{B,V}$ ) by funder:
2:   ensure  $State = \text{OPEN}$ 
3:   ensure  $c_P - \text{locked}_P \geq c_{\text{virt}}$ 
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,  $c_{\text{virt}}$ , 2,  $\text{len}(\text{hops})$ ) 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,  $\text{host}'_{\bar{P}}$ ,  $\text{host}_P$ ,  $c_P$ ,  $c_{\bar{P}}$ ,  $c_{\text{virt}}$ ,  $pk_{A,V}$ ,  $pk_{B,V}$ ,
    $(sk'_{P,F}, pk'_{P,F})$ ,  $(sk_{P,F}, pk_{P,F})$ ,  $pk_{\bar{P},F}$ ,  $pk'_{\bar{P},F}$ ,  $pk_{P,\text{out}}$ ,  $\text{len}(\text{hops})$ ) to  $\text{host}'_P$ 

12: On output (HOSTS READY,  $t_P$ ) 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,  $t_P$ ) 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:      $c_P \leftarrow c_P - c_{\text{virt}}$ 
21:     output (FUNDED,  $\text{host}_P$ , layer,  $t_P$ ) to funder // do not expect reply
   by funder
22:   end if
23:   reply (HOST ACK)

```

**Fig. 17.**

**Process LN – virtualise hops**

- 1: On (VIRTUALISING,  $\text{host}'_{\bar{P}}, pk'_{\bar{P},F}, \text{hops}, \text{fundee}, c_{\text{virt}}, i, n$ ) by  $\bar{P}$ :
- 2:   ensure  $\text{State} = \text{OPEN}$
- 3:   ensure  $c_{\bar{P}} - \text{locked}_{\bar{P}} \geq c; 1 \leq i \leq n$
- 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  $c_{\text{virt}}$  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,  $\text{host}'_P, (sk'_{P,F}, pk'_{P,F}), pk'_{\bar{P},F}, pk_{P,\text{out}}, \text{hops}[1:], \text{fundee}, c_{\text{virt}}, c_{\bar{P}}, c_P, i, n$ ) to  $\text{hops}[1].\text{left}$  and expect reply (VIRTUALISING ACK,  $\text{host\_sibling}, pk_{\text{sib},\bar{P},F}$ )
- 11:    input (INIT,  $\text{host}_P, \text{host}'_{\bar{P}}, \text{host\_sibling}, (sk'_{P,F}, pk'_{P,F}), pk'_{\bar{P},F}, pk_{\text{sib},\bar{P},F}, (sk_{P,F}, pk_{P,F}), pk_{\bar{P},F}, pk_{P,\text{out}}, c_P, c_{\bar{P}}, c_{\text{virt}}, i, t_P, \text{"left"}, n$ ) to  $\text{host}'_P$  and expect reply (HOST INIT OK)
- 12:    **else** // we are the last hop
- 13:     input (INIT,  $\text{host}_P, \text{host}'_{\bar{P}}, \text{fundee}=\text{fundee}, (sk'_{P,F}, pk'_{P,F}), pk'_{\bar{P},F}, (sk_{P,F}, pk_{P,F}), pk_{\bar{P},F}, pk_{P,\text{out}}, c_P, c_{\bar{P}}, c_{\text{virt}}, t_P, i, \text{"left"}, n$ ) 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,  $\text{host\_sibling}, (sk'_{P,F}, pk'_{P,F}), pk_{\text{sib},\bar{P},F}, pk_{\text{sib},\text{out}}, \text{hops}, \text{fundee}, c_{\text{virt}}, c_{\text{sib},\text{rem}}, c_{\text{sib}}, i, n$ ) 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,  $\text{host}'_P, pk'_{P,F}, \text{hops}, \text{fundee}, c_{\text{virt}}, i + 1, n$ ) 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,  $\text{host}_P, \text{host}'_{\bar{P}}, \text{host\_sibling}, (sk'_{P,F}, pk'_{P,F}), pk'_{\bar{P},F}, pk_{\text{sib},\bar{P},F}, (sk_{P,F}, pk_{P,F}), pk_{\bar{P},F}, pk_{\text{sib},\text{out}}, c_P, c_{\bar{P}}, c_{\text{virt}}, i, \text{"right"}, n$ ) 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:  $State \leftarrow \text{WAITING FOR COMMITMENT SIGNED}$
- 4: send (PAY,  $x$ ,  $\text{sig}_{P,C,i+1}$ ) to  $\bar{P}$
- 5: //  $P$  refers to payer and  $\bar{P}$  to payee both in local and remote code
- 6: **ensure**  $State = \text{WAITING TO GET PAID} \wedge x = y$
- 7: **if**  $\text{host}_{\bar{P}} \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$  has a **host\_sibling** **then** // we are intermediary channel
  - 8: **ensure**  $c_{\bar{P}} + x \leq c_{\bar{P},\text{sib},\text{rem}} \wedge c_{\bar{P},\text{sib}} \leq c_P - x$  // avoid value loss by griefing attack
- 9: **end if**
- 10:  $C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$  with  $x$  coins moved from  $P$ 's to  $\bar{P}$ 's output
- 11: **ensure**  $\text{VERIFY}(C_{\bar{P},i+1}, \text{sig}_{P,C,i+1}, pk_{P,F}) = \text{True}$
- 12:  $C_{P,i+1} \leftarrow C_{P,i}$  with  $x$  coins moved from  $P$ 's to  $\bar{P}$ 's output
- 13:  $\text{sig}_{\bar{P},C,i+1} \leftarrow \text{SIGN}(C_{P,i+1}, sk_{\bar{P},F})$  // kept by  $P$
- 14:  $R_{P,i} \leftarrow \text{TX}$  {input:  $C_{\bar{P},i}.\text{outputs}.P$ , output:  $(c_{\bar{P}}, pk_{P,\text{out}})$ }
- 15:  $\text{sig}_{\bar{P},R,i} \leftarrow \text{SIGN}(R_{P,i}, sk_{\bar{P},R})$
- 16:  $State \leftarrow \text{WAITING FOR PAY REVOCATION}$
- 17: **reply** (COMMITMENT SIGNED,  $\text{sig}_{\bar{P},C,i+1}$ ,  $\text{sig}_{\bar{P},R,i}$ )
- 18: **ensure**  $State = \text{WAITING FOR COMMITMENT SIGNED}$
- 19:  $C_{P,i+1} \leftarrow C_{P,i}$  with  $x$  coins moved from  $P$ 's to  $\bar{P}$ 's output

**Fig. 19.**

**Process LN.REVOCATIONSTRIP()**

```

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

```

**Fig. 20.**



**Process** LN – On (PAY,  $x$ ):

- 1: ensure  $State = \text{OPEN} \wedge c_P \geq x$
- 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  $\text{host}_P$
- 3: **end if**

**Fig. 23.**

**Process** LN – On (CHECK CHAIN FOR CLOSED):

```

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:   output (CLOSED)
10: else if  $C_{P,j} \in \Sigma \wedge C_{\bar{P},j} \in \Sigma$  then
11:    $State \leftarrow \text{CLOSED}$ 
12:   output (CLOSED)
13: 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 IF 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:   $\text{sig}'_{P,C,i} \leftarrow \text{SIGN}(C_{P,i}, sk_{P,F})$ 
19:  input (SUBMIT, ( $C_{P,i}, \text{sig}_{P,C,i}, \text{sig}'_{P,C,i}$ )) to  $\mathcal{G}_{\text{Ledger}}$ 
20: end if

```

Fig. 26.

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

```

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

```

Fig. 27.

**Process VIRT**

```

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

```

```

9:   end if

10:  // guest is trusted to give sane inputs, therefore a state machine and input
    verification are redundant
11:  On input (INIT,  $\text{host}_P$ ,  $\bar{P}$ , sibling, fundee,  $(sk_{\text{loc},\text{fund},\text{new}}, pk_{\text{loc},\text{fund},\text{new}})$ ,
     $pk_{\text{rem},\text{fund},\text{new}}$ ,  $pk_{\text{sib},\text{rem},\text{fund},\text{new}}$ ,  $(sk_{\text{loc},\text{fund},\text{old}}, pk_{\text{loc},\text{fund},\text{old}})$ ,  $pk_{\text{rem},\text{fund},\text{old}}$ ,
     $pk_{\text{loc},\text{out}}$ ,  $c_P$ ,  $c_{\bar{P}}$ ,  $c_{\text{virt}}$ ,  $t_P$ ,  $i$ , side,  $n$ ) by guest:
12:    ensure  $1 < i \leq n$  // host_funder ( $i = 1$ ) is initialised with HOST ME
13:    ensure side  $\in \{\text{"left"}, \text{"right"}\}$ 
14:    store message contents and guest // sibling,  $pk_{\text{sib},\bar{P},F}$  are missing for
    endpoints, fundee is present only in last node
15:     $(sk_{i,\text{fund},\text{new}}, pk_{i,\text{fund},\text{new}}) \leftarrow (sk_{\text{loc},\text{fund},\text{new}}, pk_{\text{loc},\text{fund},\text{new}})$ 
16:     $pk_{\text{myRem},\text{fund},\text{new}} \leftarrow pk_{\text{rem},\text{fund},\text{new}}$ 
17:    if  $i < n$  then // we are not last hop
18:       $pk_{\text{sibRem},\text{fund},\text{new}} \leftarrow pk_{\text{sib},\text{rem},\text{fund},\text{new}}$ 
19:    end if
20:    if side = "left" then
21:       $\text{side}' \leftarrow \text{"right"}$ ;  $\text{myRem} \leftarrow i - 1$ ;  $\text{sibRem} \leftarrow i + 1$ 
22:    else // side = "right"
23:       $\text{side}' \leftarrow \text{"left"}$ ;  $\text{myRem} \leftarrow i + 1$ ;  $\text{sibRem} \leftarrow i - 1$ 
24:    end if
25:     $(sk_{i,\text{side},\text{fund},\text{old}}, pk_{i,\text{side},\text{fund},\text{old}}) \leftarrow (sk_{\text{loc},\text{fund},\text{old}}, pk_{\text{loc},\text{fund},\text{old}})$ 
26:     $pk_{\text{myRem},\text{side}',\text{fund},\text{old}} \leftarrow pk_{\text{rem},\text{fund},\text{old}}$ 
27:     $pk_{i,\text{out}} \leftarrow pk_{\text{loc},\text{out}}$ 
28:     $(c_{i,\text{side}}, c_{\text{myRem},\text{side}'}, t_{i,\text{side}}) \leftarrow (c_P, c_{\bar{P}}, t_P)$ 
29:    last_poll  $\leftarrow \perp$ 
30:    if side = "left"  $\wedge i \neq n$  then
31:       $(sk_{i,j,k}, pk_{i,j,k})_{j \in \{2, \dots, n-1\}, k \in [n] \setminus \{j\}} \leftarrow \text{KEYGEN}^{(n-2)(n-1)}$ 
32:    end if
33:    output (HOST INIT OK) to guest

34:  On input (HOST ME, funder, fundee,  $\bar{P}$ ,  $\text{host}_P$ ,  $c_P$ ,  $c_{\bar{P}}$ ,  $c_{\text{virt}}$ ,  $pk_{\text{left},\text{virt}}$ ,
     $pk_{\text{right},\text{virt}}$ ,  $(sk_{1,\text{fund},\text{new}}, pk_{1,\text{fund},\text{new}})$ ,  $(sk_{1,\text{right},\text{fund},\text{old}}, pk_{1,\text{right},\text{fund},\text{old}})$ ,
     $pk_{2,\text{left},\text{fund},\text{old}}$ ,  $pk_{2,\text{left},\text{fund},\text{new}}$ ,  $pk_{1,\text{right},\text{out}}$ ,  $n$ ) by guest:
35:    last_poll  $\leftarrow \perp$ 
36:     $i \leftarrow 1$ 
37:     $c_{1,\text{right}} \leftarrow c_P$ ;  $c_{2,\text{left}} \leftarrow c_{\bar{P}}$ 
38:     $(sk_{1,j,k}, pk_{1,j,k})_{j \in \{2, \dots, n-1\}, k \in [n] \setminus \{j\}} \leftarrow \text{KEYGEN}^{(n-2)(n-1)}$ 
39:    ensure VIRT.CIRCULATEKEYSCoinsTimes() returns (OK)
40:    ensure VIRT.CIRCULATEVIRTUALSIGs() returns (OK)
41:    ensure VIRT.CIRCULATEFUNDINGSIGs() returns (OK)
42:    ensure VIRT.CIRCULATEREVOCATIONS() returns (OK)
43:    output (HOSTS READY,  $p + \sum_{j=2}^{n-1} (s - 1 + t_j)$ ) to guest //  $p$  is every how
    many blocks we have to check the chain

```

Fig. 28.

**Process** VIRT.CIRCULATEKEYSCOINSTIMES(left\_data):

```

1: if left_data is given as argument then // we are not host_funder
2:   parse left_data as  $((pk_{j,\text{fund,new}})_{j \in [i-1]}, (pk_{j,\text{left,fund,old}})_{j \in \{2,\dots,i-1\}},$ 
    $(pk_{j,\text{right,fund,old}})_{j \in [i-1]}, (pk_{j,\text{out}})_{j \in [i-1]}, (c_{j,\text{left}})_{j \in \{2,\dots,i-1\}}, (c_{j,\text{right}})_{j \in [i-1]},$ 
    $(t_j)_{j \in [i-1]}, pk_{\text{left,virt}}, pk_{\text{right,virt}}, (pk_{h,j,k})_{h \in [i-1], j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}})$ 
3:   if we have a sibling then // we are not host_fundee
4:     input (KEYS AND COINS FORWARD, (left_data,  $(sk_{i,\text{left,fund,old}},$ 
    $pk_{i,\text{left,fund,old}}), c_{i,\text{left}}, t_{i,\text{left}}, (sk_{i,j,k}, pk_{i,j,k})_{j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}})$  to sibling
5:     store input as left_data and parse it as  $((pk_{j,\text{fund,new}})_{j \in [i-1]},$ 
    $(pk_{j,\text{left,fund,old}})_{j \in \{2,\dots,i\}}, (pk_{j,\text{right,fund,old}})_{j \in [i-1]}, (pk_{j,\text{out}})_{j \in [i-1]},$ 
    $(c_{j,\text{left}})_{j \in \{2,\dots,i\}}, (c_{j,\text{right}})_{j \in [i-1]}, (t_j)_{j \in [i-1]}, sk_{i,\text{left,fund,old}}, t_{i,\text{left}}, pk_{\text{left,virt}},$ 
    $pk_{\text{right,virt}}, (pk_{h,j,k})_{h \in [i], j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}}, (sk_{i,j,k})_{j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}})$ 
6:      $t_i \leftarrow \max(t_{i,\text{left}}, t_{i,\text{right}})$ 
7:     replace  $t_{i,\text{left}}$  in left_data with  $t_i$ 
8:     remove  $sk_{i,\text{left,fund,old}}$  and  $(sk_{i,j,k})_{j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}}$  from left_data
9:     call VIRT.CIRCULATEKEYSCOINSTIMES(left_data) of  $\bar{P}$  and assign
   returned value to right_data
10:    parse right_data as  $((pk_{j,\text{fund,new}})_{j \in \{i+1,\dots,n\}},$ 
    $(pk_{j,\text{left,fund,old}})_{j \in \{i+1,\dots,n\}}, (pk_{j,\text{right,fund,old}})_{j \in \{i+1,\dots,n-1\}}, (pk_{j,\text{out}})_{j \in \{i+1,\dots,n\}},$ 
    $(c_{j,\text{left}})_{j \in \{i+1,\dots,n\}}, (c_{j,\text{right}})_{j \in \{i+1,\dots,n-1\}}, (t_j)_{j \in \{i+1,\dots,n\}},$ 
    $(pk_{h,j,k})_{h \in \{i+1,\dots,n\}, j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}})$ 
11:    output (KEYS AND COINS BACK, right_data,  $(sk_{i,\text{right,fund,old}},$ 
    $pk_{i,\text{right,fund,old}}), c_{i,\text{right}}, t_i)$ 
12:    store output as right_data and parse it as  $((pk_{j,\text{fund,new}})_{j \in \{i+1,\dots,n\}},$ 
    $(pk_{j,\text{left,fund,old}})_{j \in \{i+1,\dots,n\}}, (pk_{j,\text{right,fund,old}})_{j \in \{i,\dots,n-1\}}, (pk_{j,\text{out}})_{j \in \{i+1,\dots,n\}},$ 
    $(c_{j,\text{left}})_{j \in \{i+1,\dots,n\}}, (c_{j,\text{right}})_{j \in \{i,\dots,n-1\}}, (t_j)_{j \in \{i,\dots,n\}},$ 
    $(pk_{h,j,k})_{h \in \{i+1,\dots,n\}, j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}}, sk_{i,\text{right,fund,old}})$ 
13:    remove  $sk_{i,\text{right,fund,old}}$  from right_data
14:    return (right_data,  $pk_{i,\text{fund,new}}, pk_{i,\text{left,fund,old}}, pk_{i,\text{out}}, c_{i,\text{left}})$ 
15:  else // we are host_fundee
16:    output (CHECK KEYS,  $(pk_{\text{left,virt}}, pk_{\text{right,virt}})$ ) to fundee and expect
   reply (KEYS OK)
17:    return  $(pk_{n,\text{fund,new}}, pk_{n,\text{left,fund,old}}, pk_{n,\text{out}}, c_{n,\text{left}}, t_n)$ 
18:  end if
19: else // we are host_funder
20:   call VIRT.CIRCULATEKEYSCOINSTIMES( $pk_{1,\text{fund,new}}, pk_{1,\text{right,fund,old}}, pk_{1,\text{out}},$ 
    $c_{1,\text{right}}, t_1, pk_{\text{left,virt}}, pk_{\text{right,virt}}, (pk_{1,j,k})_{j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}})$  of  $\bar{P}$  and assign
   returned value to right_data
21:   parse right_data as  $((pk_{j,\text{fund,new}})_{j \in \{2,\dots,n\}}, (pk_{j,\text{left,fund,old}})_{j \in \{2,\dots,n\}},$ 
    $(pk_{j,\text{right,fund,old}})_{j \in \{2,\dots,n-1\}}, (pk_{j,\text{out}})_{j \in \{2,\dots,n\}}, (c_{j,\text{left}})_{j \in \{2,\dots,n\}},$ 
    $(c_{j,\text{right}})_{j \in \{2,\dots,n-1\}}, (t_j)_{j \in \{2,\dots,n\}}, (pk_{h,j,k})_{h \in \{2,\dots,n\}, j \in \{2,\dots,n-1\}, k \in [n] \setminus \{j\}})$ 
22:   return (OK)
23: end if

```

Fig. 29.

# Process VIRT

```

1: GETMIDTXS( $i, n, c_{\text{virt}}, c_{\text{rem},\text{left}}, c_{\text{loc},\text{left}}, c_{\text{loc},\text{right}}, c_{\text{rem},\text{right}}, pk_{\text{rem},\text{left},\text{fund},\text{old}},$ 
 $pk_{\text{loc},\text{left},\text{fund},\text{old}}, pk_{\text{loc},\text{right},\text{fund},\text{old}}, pk_{\text{rem},\text{right},\text{fund},\text{old}}, pk_{\text{rem},\text{left},\text{fund},\text{new}},$ 
 $pk_{\text{loc},\text{left},\text{fund},\text{new}}, pk_{\text{loc},\text{right},\text{fund},\text{new}}, pk_{\text{rem},\text{right},\text{fund},\text{new}}, pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}},$ 
 $pk_{\text{loc},\text{out}}, (pk_{p,j,k})_{p \in [n], j \in [n-1] \setminus \{1\}, k \in [n-1] \setminus \{1,j\}}, (pk_{p,2,1})_{p \in [n]}, (pk_{p,n-1,n})_{p \in [n]},$ 
 $(t_j)_{j \in [n-1] \setminus \{1\}})$ :
2:   ensure  $1 < i < n$ 
3:   ensure  $c_{\text{rem},\text{left}} \geq c_{\text{virt}} \wedge c_{\text{loc},\text{left}} \geq c_{\text{virt}}$  // left parties fund virtual channel
4:   ensure  $c_{\text{rem},\text{left}} \geq c_{\text{loc},\text{right}} \wedge c_{\text{rem},\text{right}} \geq c_{\text{loc},\text{left}}$  // avoid griefing attack
5:    $c_{\text{left}} \leftarrow c_{\text{rem},\text{left}} + c_{\text{loc},\text{left}}; c_{\text{right}} \leftarrow c_{\text{loc},\text{right}} + c_{\text{rem},\text{right}}$ 
6:    $\text{left\_old\_fund} \leftarrow 2 / \{pk_{\text{rem},\text{left},\text{fund},\text{old}}, pk_{\text{loc},\text{left},\text{fund},\text{old}}\}$ 
7:    $\text{right\_old\_fund} \leftarrow 2 / \{pk_{\text{loc},\text{right},\text{fund},\text{old}}, pk_{\text{rem},\text{right},\text{fund},\text{old}}\}$ 
8:    $\text{left\_new\_fund} \leftarrow 2 / \{pk_{\text{rem},\text{left},\text{fund},\text{new}}, pk_{\text{loc},\text{left},\text{fund},\text{new}}\}$ 
9:    $\text{right\_new\_fund} \leftarrow 2 / \{pk_{\text{loc},\text{right},\text{fund},\text{new}}, pk_{\text{rem},\text{right},\text{fund},\text{new}}\}$ 
10:   $\text{virt\_fund} \leftarrow 2 / \{pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}}\}$ 
11:  for all  $j \in [n-1] \setminus \{1\}, k \in [n-1] \setminus \{1,j\}$  do
12:     $\text{all}_{j,k} \leftarrow n / \{pk_{1,j,k}, \dots, pk_{n,j,k}\} \wedge "k"$ 
13:  end for
14:  if  $i = 2$  then
15:     $\text{all}_{2,1} \leftarrow n / \{pk_{1,2,1}, \dots, pk_{n,2,1}\} \wedge "1"$ 
16:  end if
17:  if  $i = n-1$  then
18:     $\text{all}_{n-1,n} \leftarrow n / \{pk_{1,n-1,n}, \dots, pk_{n,n-1,n}\} \wedge "n"$ 
19:  end if
20:  // After funding is complete,  $A_j$  has the signature of all other parties for
  all  $\text{all}_{j,k}$  inputs, but other parties do not have  $A_j$ 's signature for this input,
  therefore only  $A_j$  can publish it.
21:  //  $\text{TX}_{i,j,k} := i$ -th move,  $j, k$  input interval start and end.  $j, k$  unneeded for
 $i = 1, k$  unneeded for  $i = 2$ .
22:   $\text{TX}_1 \leftarrow \text{TX}$ :
23:    inputs:
24:      ( $c_{\text{left}}, \text{left\_old\_fund}$ ),
25:      ( $c_{\text{right}}, \text{right\_old\_fund}$ )
26:    outputs:
27:      ( $c_{\text{left}} - c_{\text{virt}}, \text{left\_new\_fund}$ ),
28:      ( $c_{\text{right}} - c_{\text{virt}}, \text{right\_new\_fund}$ ),
29:      ( $c_{\text{virt}}, pk_{\text{loc},\text{out}}$ ),
30:      ( $c_{\text{virt}},$ 
31:        (if  $(i-1 > 1)$  then  $\text{all}_{i-1,i}$  else False)
32:         $\vee$  (if  $(i+1 < n)$  then  $\text{all}_{i+1,i}$  else False)
33:         $\vee$  (
34:          if  $(i-1 = 1 \wedge i+1 = n)$  then  $\text{virt\_fund}$ 
35:          else if  $(i-1 > 1 \wedge i+1 = n)$  then  $\text{virt\_fund} + t_{i-1}$ 
36:          else if  $(i-1 = 1 \wedge i+1 < n)$  then  $\text{virt\_fund} + t_{i+1}$ 

```

```

37:         else /* $i - 1 > 1 \wedge i + 1 < n$ */ virt_fund + max( $t_{i-1}, t_{i+1}$ )
38:     )
39: )
40: if  $i = 2$  then
41:     TX2,1 ← TX:
42:     inputs:
43:         ( $c_{\text{virt}}, \text{all}_{2,1}$ ),
44:         ( $c_{\text{right}}, \text{right\_old\_fund}$ )
45:     outputs:
46:         ( $c_{\text{right}} - c_{\text{virt}}, \text{right\_new\_fund}$ ),
47:         ( $c_{\text{virt}}, pk_{\text{loc,out}}$ ),
48:         ( $c_{\text{virt}},$ 
49:             if ( $n > 3$ ) then ( $\text{all}_{3,2} \vee (\text{virt\_fund} + t_3)$ )
50:             else virt_fund
51:         )
52: end if
53: if  $i = n - 1$  then
54:     TX2,n ← TX:
55:     inputs:
56:         ( $c_{\text{left}}, \text{left\_old\_fund}$ ),
57:         ( $c_{\text{virt}}, \text{all}_{n-1,n}$ )
58:     outputs:
59:         ( $c_{\text{left}} - c_{\text{virt}}, \text{left\_new\_fund}$ ),
60:         ( $c_{\text{virt}}, pk_{\text{loc,out}}$ ),
61:         ( $c_{\text{virt}},$ 
62:             if ( $n - 2 > 1$ ) then ( $\text{all}_{n-2,n-1} \vee (\text{virt\_fund} + t_{n-2})$ )
63:             else virt_fund
64:         )
65: end if
66: for all  $k \in \{2, \dots, i - 1\}$  do //  $i - 2$  txs
67:     TX2,k ← TX:
68:     inputs:
69:         ( $c_{\text{virt}}, \text{all}_{i,k}$ ),
70:         ( $c_{\text{right}}, \text{right\_old\_fund}$ )
71:     outputs:
72:         ( $c_{\text{right}} - c_{\text{virt}}, \text{right\_new\_fund}$ ),
73:         ( $c_{\text{virt}}, pk_{\text{loc,out}}$ ),
74:         ( $c_{\text{virt}},$ 
75:             (if ( $k - 1 > 1$ ) then  $\text{all}_{k-1,i}$  else False)
76:              $\vee$  (if ( $i + 1 < n$ ) then  $\text{all}_{i+1,k}$  else False)
77:              $\vee$  (
78:                 if ( $k - 1 = 1 \wedge i + 1 = n$ ) then virt_fund
79:                 else if ( $k - 1 > 1 \wedge i + 1 = n$ ) then virt_fund +  $t_{k-1}$ 

```

```

80:             else if  $(k - 1 = 1 \wedge i + 1 < n)$  then virt_fund +  $t_{i+1}$ 
81:             else  $/^*k - 1 > 1 \wedge i + 1 < n^*/$  virt_fund +  $\max(t_{k-1}, t_{i+1})$ 
82:         )
83:     )
84: end for
85: for all  $k \in \{i + 1, \dots, n - 1\}$  do  $// n - i - 1$  txs
86:      $\text{TX}_{2,k} \leftarrow \text{TX}$ :
87:     inputs:
88:         ( $c_{\text{left}}$ , left_old_fund)
89:         ( $c_{\text{virt}}$ ,  $\text{all}_{i,k}$ ),
90:     outputs:
91:         ( $c_{\text{left}} - c_{\text{virt}}$ , left_new_fund),
92:         ( $c_{\text{virt}}$ ,  $pk_{\text{loc,out}}$ ),
93:         ( $c_{\text{virt}}$ ,
94:             (if  $(i - 1 > 1)$  then  $\text{all}_{i-1,k}$  else False)
95:              $\vee$  (if  $(k + 1 < n)$  then  $\text{all}_{k+1,i}$  else False)
96:              $\vee$  (
97:                 if  $(i - 1 = 1 \wedge k + 1 = n)$  then virt_fund
98:                 else if  $(i - 1 > 1 \wedge k + 1 = n)$  then virt_fund +  $t_{i-1}$ 
99:                 else if  $(i - 1 = 1 \wedge k + 1 < n)$  then virt_fund +  $t_{k+1}$ 
100:                 else  $/^*i - 1 > 1 \wedge k + 1 < n^*/$ 
101:                 virt_fund +  $\max(t_{i-1}, t_{k+1})$ 
102:             )
103:         )
104:     )
105: end for
106: if  $i = 2$  then  $m \leftarrow 1$  else  $m \leftarrow 2$ 
107: if  $i = n - 1$  then  $l \leftarrow n$  else  $l \leftarrow n - 1$ 
108: for all  $(k_1, k_2) \in \{m, \dots, i - 1\} \times \{i + 1, \dots, l\}$  do  $// (i - m) \cdot (l - i)$  txs
109:      $\text{TX}_{3,k_1,k_2} \leftarrow \text{TX}$ :
110:     inputs:
111:         ( $c_{\text{virt}}$ ,  $\text{all}_{i,k_1}$ ),
112:         ( $c_{\text{virt}}$ ,  $\text{all}_{i,k_2}$ )
113:     outputs:
114:         ( $c_{\text{virt}}$ ,  $pk_{\text{loc,out}}$ ),
115:         ( $c_{\text{virt}}$ ,
116:             (if  $(k_1 - 1 > 1)$  then  $\text{all}_{k_1-1, \min(k_2, n-1)}$  else False)
117:              $\vee$  (if  $(k_2 + 1 < n)$  then  $\text{all}_{k_2+1, \max(k_1, 2)}$  else False)
118:              $\vee$  (
119:                 if  $(k_1 - 1 \leq 1 \wedge k_2 + 1 \geq n)$  then virt_fund
120:                 else if  $(k_1 - 1 > 1 \wedge k_2 + 1 \geq n)$  then virt_fund +  $t_{k_1-1}$ 
121:                 else if  $(k_1 - 1 \leq 1 \wedge k_2 + 1 < n)$  then virt_fund +  $t_{k_2+1}$ 
122:                 else  $/^*k_1 - 1 > 1 \wedge k_2 + 1 < n^*/$ 
123:                 virt_fund +  $\max(t_{k_1-1}, t_{k_2+1})$ 
124:             )
125:         )
126:     )
127: end for

```



```

123:         )
124:     end for
125:     return (
126:         TX1,
127:         (TX2,k)k ∈ {m,...,l} \ {i},
128:         (TX3,k1,k2)(k1,k2) ∈ {m,...,i-1} × {i+1,...,l}
129:     )

```

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: GETENDPOINTTX(i, n, cvirt, cleft, cright, pkleft,fund,old, pkright,fund,old,
   pkleft,fund,new, pkright,fund,new, pkleft,virt, pkright,virt, (pkall,j)j ∈ [n], t):
3:     ensure i ∈ {1, n}
4:     ensure cleft ≥ cvirt // left party funds virtual channel
5:     ctot ← cleft + cright
6:     old_fund ← 2/{pkleft,fund,old, pkright,fund,old}
7:     new_fund ← 2/{pkleft,fund,new, pkright,fund,new}
8:     virt_fund ← 2/{pkleft,virt, pkright,virt}
9:     if i = 1 then // funder's tx
10:         all ← n/{pkall,1, ..., pkall,n} ∧ "1"
11:     else // fundee's tx
12:         all ← n/{pkall,1, ..., pkall,n} ∧ "n"
13:     end if
14:     TX1 ← TX: // endpoints only have an "initiator" tx
15:     inputs:
16:         (ctot, old_fund)
17:     outputs:
18:         (ctot - cvirt, new_fund),
19:         (cvirt, all ∨ (virt_fund + t))
20:     return TX1

```

Fig. 31.

**Process** VIRT.SIBLINGSIGS()

```

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

5: // notation:  $\text{sig}(TX, pk) := \text{sig}$  with ANYPREVOUT flag such that
    $\text{VERIFY}(TX, \text{sig}, pk) = \text{True}$ 
6: ensure that the following signatures are present in sigsbyLeft and store them:
   - //  $(l - m) \cdot (i - 1)$  signatures
7:    $\forall k \in \{m, \dots, l\} \setminus \{i\}, \forall j \in [i - 1] :$ 
8:      $\text{sig}(TX_{i,2,k}, pk_{j,i,k})$ 
   - //  $2 \cdot (i - m) \cdot (l - i) \cdot (i - 1)$  signatures
9:    $\forall k_1 \in \{m, \dots, i - 1\}, \forall k_2 \in \{i + 1, \dots, l\}, \forall j \in [i - 1] :$ 
10:     $\text{sig}(TX_{i,3,k_1,k_2}, pk_{j,i,k_1}), \text{sig}(TX_{i,3,k_1,k_2}, pk_{j,i,k_2})$ 
11: sigstoRight  $\leftarrow$  sigsbyLeft

12: for all  $j \in \{2, \dots, n - 1\} \setminus \{i\}$  do
13:   if  $j = 2$  then  $m' \leftarrow 1$  else  $m' \leftarrow 2$ 
14:   if  $j = n - 1$  then  $l' \leftarrow n$  else  $l' \leftarrow n - 1$ 
15:    $(TX_{j,1}, (TX_{j,2,k})_{k \in \{m', \dots, l'\} \setminus \{j\}}, (TX_{j,3,k_1,k_2})_{(k_1,k_2) \in \{m', \dots, i-1\} \setminus \{i+1, \dots, l'\}}) \leftarrow$ 
     GETMIDTXS( $j, n, c_{virt}, c_{j-1, \text{right}}, c_{j, \text{left}}, c_{j, \text{right}}, c_{j+1, \text{left}}, pk_{j-1, \text{right}, \text{fund}, \text{old}},$ 
      $pk_{j, \text{left}, \text{fund}, \text{old}}, pk_{j, \text{right}, \text{fund}, \text{old}}, pk_{j+1, \text{left}, \text{fund}, \text{old}}, pk_{j-1, \text{fund}, \text{new}}, pk_{j, \text{fund}, \text{new}},$ 
      $pk_{j, \text{fund}, \text{new}}, pk_{j+1, \text{fund}, \text{new}}, pk_{i, \text{left}, \text{virt}}, pk_{i, \text{right}, \text{virt}}, pk_{j, \text{out}},$ 
      $(pk_{k,p,s})_{k \in [n], p \in [n-1] \setminus \{1\}, s \in [n-1] \setminus \{1, p\}}, (pk_{k,2,1})_{k \in [n]}, (pk_{k,n-1,n})_{k \in [n]},$ 
      $(t_k)_{k \in [n-1] \setminus \{1\}})$ 
16:   if  $j < i$  then sigs  $\leftarrow$  sigstoLeft else sigs  $\leftarrow$  sigstoRight
17:   for all  $k \in \{m', \dots, l'\} \setminus \{j\}$  do
18:     add SIGN( $TX_{j,2,k}, sk_{i,j,k}, \text{ANYPREVOUT}$ ) to sigs
19:   end for
20:   for all  $k_1 \in \{m', \dots, j - 1\}, k_2 \in \{j + 1, \dots, l'\}$  do
21:     add SIGN( $TX_{j,3,k_1,k_2}, sk_{i,j,k_1}, \text{ANYPREVOUT}$ ) to sigs
22:     add SIGN( $TX_{j,3,k_1,k_2}, sk_{i,j,k_2}, \text{ANYPREVOUT}$ ) to sigs
23:   end for
24: end for
25: if  $i + 1 = n$  then // next hop is host_fundee
26:    $TX_{n,1} \leftarrow \text{VIRT.GETENDPOINTTX}(n, n, c_{virt}, c_{n-1, \text{right}}, c_{n, \text{left}},$ 
      $pk_{n-1, \text{right}, \text{fund}, \text{old}}, pk_{n, \text{left}, \text{fund}, \text{old}}, pk_{n-1, \text{fund}, \text{new}}, pk_{n, \text{fund}, \text{new}}, pk_{i, \text{left}, \text{virt}},$ 
      $pk_{i, \text{right}, \text{virt}}, (pk_{j,n-1,n})_{j \in [n]}, t_{n-1})$ 
27: end if

```

```

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

```

Fig. 32.

**Process**  $\text{VIRT.INTERMEDIARYSIGS}()$

```

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

```

Fig. 33.

**Process**  $\text{VIRT.HOSTFUNDEESIGS}()$

```

1:  $\text{TX}_{n,1} \leftarrow \text{VIRT.GETENDPOINTTX}(n, n, c_{\text{virt}}, c_{n-1,\text{right}}, c_{n,\text{left}},$ 
    $pk_{n-1,\text{right},\text{fund},\text{old}}, pk_{n,\text{right},\text{fund},\text{old}}, pk_{n-1,\text{fund},\text{new}}, pk_{n,\text{fund},\text{new}}, pk_{i,\text{left},\text{virt}},$ 
    $pk_{i,\text{right},\text{virt}}, (pk_{j,n-1,n})_{j \in [n]}, t_{n-1})$ 
2: for all  $j \in [n - 1] \setminus \{1\}$  do

```

```

3:   if  $j = 2$  then  $m \leftarrow 1$  else  $m \leftarrow 2$ 
4:   if  $j = n - 1$  then  $l \leftarrow n$  else  $l \leftarrow n - 1$ 
5:    $(TX_{j,1}, (TX_{j,2,k})_{k \in \{m, \dots, l\} \setminus \{j\}}, (TX_{i,3,k_1,k_2})_{(k_1,k_2) \in \{m, \dots, i-1\} \setminus \{i+1, \dots, l\}}) \leftarrow$ 
   VIRT.GETMIDTXS( $j, n, c_{\text{virt}}, c_{j-1, \text{right}}, c_{j, \text{left}}, c_{j, \text{right}}, c_{j+1, \text{left}},$ 
    $pk_{j-1, \text{right}, \text{fund}, \text{old}}, pk_{j, \text{left}, \text{fund}, \text{old}}, pk_{j, \text{right}, \text{fund}, \text{old}}, pk_{j+1, \text{left}, \text{fund}, \text{old}},$ 
    $pk_{j-1, \text{fund}, \text{new}}, pk_{j, \text{fund}, \text{new}}, pk_{j, \text{fund}, \text{new}}, pk_{j+1, \text{fund}, \text{new}}, pk_{\text{left}, \text{virt}}, pk_{\text{right}, \text{virt}},$ 
    $pk_{j, \text{out}}, (pk_{j,s,k})_{j \in [n], s \in [n-1] \setminus \{1\}, k \in [n-1] \setminus \{1, s\}}, (pk_{j,2,1})_{j \in [n]}, (pk_{j,n-1,n})_{j \in [n]},$ 
    $(t_j)_{j \in [n-1] \setminus \{1\}})$ 
6:    $\text{sig}_{\text{toLeft}} \leftarrow \emptyset$ 
7:   for all  $k \in \{m, \dots, l\} \setminus \{j\}$  do
8:     add SIGN( $TX_{j,2,k}, sk_{n,j,k}, \text{ANYPREVOUT}$ ) to  $\text{sig}_{\text{toLeft}}$ 
9:   end for
10:  for all  $k_1 \in \{m, \dots, j-1\}, k_2 \in \{j+1, \dots, l\}$  do
11:    add SIGN( $TX_{j,3,k_1,k_2}, sk_{n,j,k_1}, \text{ANYPREVOUT}$ ) to  $\text{sig}_{\text{toLeft}}$ 
12:    add SIGN( $TX_{j,3,k_1,k_2}, sk_{n,j,k_2}, \text{ANYPREVOUT}$ ) to  $\text{sig}_{\text{toLeft}}$ 
13:  end for
14: end for
15: return  $\text{sig}_{\text{toLeft}}$ 

```

Fig. 34.

**Process** VIRT.HOSTFUNDERSIGS()

```

1: for all  $j \in [n-1] \setminus \{1\}$  do
2:   if  $j = 2$  then  $m \leftarrow 1$  else  $m \leftarrow 2$ 
3:   if  $j = n - 1$  then  $l \leftarrow 1$  else  $l \leftarrow 2$ 
4:    $(TX_{j,1}, (TX_{j,2,k})_{k \in \{m, \dots, l\} \setminus \{j\}}, (TX_{i,3,k_1,k_2})_{(k_1,k_2) \in \{m, \dots, i-1\} \setminus \{i+1, \dots, l\}}) \leftarrow$ 
   VIRT.GETMIDTXS( $j, n, c_{\text{virt}}, c_{j-1, \text{right}}, c_{j, \text{left}}, c_{j, \text{right}}, c_{j+1, \text{left}},$ 
    $pk_{j-1, \text{right}, \text{fund}, \text{old}}, pk_{j, \text{left}, \text{fund}, \text{old}}, pk_{j, \text{right}, \text{fund}, \text{old}}, pk_{j+1, \text{left}, \text{fund}, \text{old}},$ 
    $pk_{j-1, \text{fund}, \text{new}}, pk_{j, \text{fund}, \text{new}}, pk_{j, \text{fund}, \text{new}}, pk_{j+1, \text{fund}, \text{new}}, pk_{\text{left}, \text{virt}}, pk_{\text{right}, \text{virt}},$ 
    $pk_{j, \text{out}}, (pk_{j,s,k})_{j \in [n], s \in [n-1] \setminus \{1\}, k \in [n-1] \setminus \{1, s\}}, (pk_{j,2,1})_{j \in [n]}, (pk_{j,n-1,n})_{j \in [n]},$ 
    $(t_j)_{j \in [n-1] \setminus \{1\}})$ 
5:    $\text{sig}_{\text{toRight}} \leftarrow \emptyset$ 
6:   for all  $k \in \{m, \dots, l\} \setminus \{j\}$  do
7:     add SIGN( $TX_{j,2,k}, sk_{1,j,k}, \text{ANYPREVOUT}$ ) to  $\text{sig}_{\text{toRight}}$ 
8:   end for
9:   for all  $k_1 \in \{m, \dots, j-1\}, k_2 \in \{j+1, \dots, l\}$  do
10:    add SIGN( $TX_{j,3,k_1,k_2}, sk_{1,j,k_1}, \text{ANYPREVOUT}$ ) to  $\text{sig}_{\text{toRight}}$ 
11:    add SIGN( $TX_{j,3,k_1,k_2}, sk_{1,j,k_2}, \text{ANYPREVOUT}$ ) to  $\text{sig}_{\text{toRight}}$ 
12:   end for
13: end for
14: call VIRT.CIRCULATEVIRTUALSIGS( $\text{sig}_{\text{toRight}}$ ) of  $\bar{P}$  and assign output to
    $\text{sig}_{\text{byRight}}$ 
15:  $TX_{1,1} \leftarrow \text{VIRT.GETENDPOINTTX}(1, n, c_{\text{virt}}, c_{1, \text{right}}, c_{2, \text{left}}, pk_{1, \text{right}, \text{fund}, \text{old}},$ 
    $pk_{2, \text{left}, \text{fund}, \text{old}}, pk_{1, \text{fund}, \text{new}}, pk_{2, \text{fund}, \text{new}}, pk_{\text{left}, \text{virt}}, pk_{\text{right}, \text{virt}}, (pk_{j,2,1})_{j \in [n]}, t_2)$ 

```

```
16: return (OK)
```

Fig. 35.

**Process** VIRT.CIRCULATEVIRTUALSIGS(sigs<sub>byLeft</sub>)

```
1: if 1 < i < n then // we are not host_funder nor host_fundee
2:   return VIRT.INTERMEDIARYSIGS()
3: else if i = 1 then // we are host_funder
4:   return VIRT.HOSTFUNDERSIGS()
5: else if i = n then // we are host_fundee
6:   return VIRT.HOSTFUNDEESIGS()
7: end if // it is always 1 ≤ i ≤ n - c.f. Fig. 28, l. 12 and l. 37
```

Fig. 36.

**Process** VIRT.CIRCULATEFUNDINGSIGS(sigs<sub>byLeft</sub>)

```
1: if 1 < i < n then // we are not endpoint
2:   if i = 2 then m ← 1 else m ← 2
3:   if i = n - 1 then l ← n else l ← n - 1
4:   ensure that the following signatures are present in sigsbyLeft and store them:
   - // 1 signature
5:     sig(TXi,1, pki-1,right,fund,old)
   - // n - 3 + χi=2 + χi=n-1 signatures
6:     ∀k ∈ {m, ..., l} \ {i}
7:     sig(TXi,2,k, pki-1,right,fund,old)
8:   input (VIRTUAL BASE SIG FORWARD, sigsbyLeft) to sibling
9:   extract and store sig(TXi,1, pki-1,right,fund,old) and ∀k ∈ {m, ..., l} \ {i}
   sig(TXi,2,k, pki-1,right,fund,old) from sigsbyLeft // same signatures as sibling
10:  sigstoRight ← {SIGN(TXi+1,1, ski,right,fund,old, ANYPREVOUT)}
11:  if i + 1 < n then
12:    if i + 1 = n - 1 then l' ← n else l' ← n - 1
13:    for all k ∈ {2, ..., l'} do
14:      add SIGN(TXi+1,2,k, ski,right,fund,old, ANYPREVOUT) to sigstoRight
15:    end for
16:  end if
17:  call VIRT.CIRCULATEFUNDINGSIGS(sigstoRight) of P̄ and assign returned
  values to sigsbyRight
18:  ensure that the following signatures are present in sigsbyRight and store
  them:
```

```

- // 1 signature
19:   sig(TXi,1, pki+1,left,fund,old)
- // n - 3 + χi=2 + χi=n-1 signatures
20:   ∀k ∈ {m, ..., l} \ {i}
21:   sig(TXi,2,k, pki+1,right,fund,old)
22:   output (VIRTUAL BASE SIG BACK, sigsbyRight)
23:   extract and store sig(TXi,1, pki+1,right,fund,old) and ∀k ∈ {m, ..., l} \ {i}
   sig(TXi,2,k, pki+1,right,fund,old) from sigsbyRight // same signatures as sibling
24:   sigtoLeft ← {SIGN(TXi-1,1, ski,left,fund,old, ANYPREVOUT)}
25:   if i - 1 > 1 then
26:     if i - 1 = 2 then m' ← 1 else m' ← 2
27:     for all k ∈ {m', ..., n - 1} do
28:       add SIGN(TXi-1,2,k, ski,left,fund,old, ANYPREVOUT) to sigstoLeft
29:     end for
30:   end if
31:   return sigstoLeft
32: else if i = 1 then // we are host_funder
33:   sigstoRight ← {SIGN(TX2,1, sk1,right,fund,old, ANYPREVOUT)}
34:   if 2 = n - 1 then l' ← n else l' ← n - 1
35:   for all k ∈ {3, ..., l'} do
36:     add SIGN(TX2,2,k, sk1,right,fund,old, ANYPREVOUT) to sigstoRight
37:   end for
38:   call VIRT.CIRCULATEFUNDINGSIGS(sigstoRight) of  $\bar{P}$  and assign returned
   value to sigsbyRight
39:   ensure that sig(TX1,1, pk2,left,fund,old) is present in sigsbyRight and store it
40:   return (OK)
41: else if i = n then // we are host_fundee
42:   ensure sig(TXn,1, pkn-1,right,fund,old) is present in sigsbyLeft and store it
43:   sigstoLeft ← {SIGN(TXn-1,1, skn,left,fund,old, ANYPREVOUT)}
44:   if n - 1 = 2 then m' ← 1 else m' ← 2
45:   for all k ∈ {m', ..., n - 2} do
46:     add SIGN(TXn-1,2,k, skn,left,fund,old, ANYPREVOUT) to sigstoLeft
47:   end for
48:   return sigstoLeft
49: end if // it is always 1 ≤ i ≤ n - c.f. Fig. 28, l. 12 and l. 37

```

Fig. 37.

**Process** VIRT.CIRCULATEREVOCATIONS(revoc\_by\_prev)

```

1: if revoc_by_prev is given as argument then // we are not host_funder
2:   ensure guest.PROCESSREMOTEREVOCATION(revoc_by_prev) returns (OK)
3: else // we are host_funder
4:   revoc_for_next ← guest.REVOKEPREVIOUS()

```

```

5:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
6:    $\text{last\_poll} \leftarrow |\Sigma|$ 
7:   call  $\text{VIRT.CIRCULATEREVOCATIONS}(\text{revoc\_for\_next})$  of  $\bar{P}$  and assign
   returned value to  $\text{revoc\_by\_next}$ 
8:   ensure  $\text{guest.PROCESSREMOTEREVOCATION}(\text{revoc\_by\_next})$  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:    $\text{revoc\_for\_next} \leftarrow \text{guest.REVOKEPREVIOUS}()$ 
14:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
15:    $\text{last\_poll} \leftarrow |\Sigma|$ 
16:   call  $\text{VIRT.CIRCULATEREVOCATIONS}(\text{revoc\_for\_next})$  of  $\bar{P}$  and assign
   output to  $\text{revoc\_by\_next}$ 
17:   ensure  $\text{guest.PROCESSREMOTEREVOCATION}(\text{revoc\_by\_next})$  returns (OK)
18:   output (HOSTS READY,  $t_i$ ) to guest and expect reply (HOST ACK)
19:   output (VIRTUAL REVOCATION BACK)
20: end if
21:  $\text{revoc\_for\_prev} \leftarrow \text{guest.REVOKEPREVIOUS}()$ 
22: if  $1 < i < n$  then // we are intermediary
23:   output (HOSTS READY,  $t_i$ ) to guest and expect reply (HOST ACK) //  $p$  is
   every how many blocks we have to check the chain
24: else // we are host_fundee, case of host_funder covered earlier
25:   output (HOSTS READY,  $p + \sum_{j=2}^{n-1} (s - 1 + t_j)$ ) to guest and expect reply
   (HOST ACK)
26: end if
27: return  $\text{revoc\_for\_prev}$ 

```

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:    $k_1 \leftarrow 0$ 
4:   if  $\text{TX}_{i-1,1}$  is defined and  $\text{TX}_{i-1,1} \in \Sigma$  then
5:      $k_1 \leftarrow i - 1$ 
6:   end if
7:   for all  $k \in [i - 2]$  do
8:     if  $\text{TX}_{i-1,2,k}$  is defined and  $\text{TX}_{i-1,2,k} \in \Sigma$  then
9:        $k_1 \leftarrow k$ 
10:    end if

```

```

11:   end for
12:    $k_2 \leftarrow 0$ 
13:   if  $\text{TX}_{i+1,1}$  is defined and  $\text{TX}_{i+1,1} \in \Sigma$  then
14:      $k_2 \leftarrow i + 1$ 
15:   end if
16:   for all  $k \in \{i + 2, \dots, n\}$  do
17:     if  $\text{TX}_{i+1,2,k}$  is defined and  $\text{TX}_{i+1,2,k} \in \Sigma$  then
18:        $k_2 \leftarrow k$ 
19:     end if
20:   end for
21:    $\text{last\_poll} \leftarrow |\Sigma|$ 
22:   if  $k_1 > 0 \vee k_2 > 0$  then // at least one neighbour has published its TX
23:     ignore all messages except for (CHECK IF CLOSING) by  $R$ 
24:      $\text{State} \leftarrow \text{CLOSING}$ 
25:      $\text{sigs} \leftarrow \emptyset$ 
26:   end if
27:   if  $k_1 > 0 \wedge k_2 > 0$  then // both neighbours have published their TXs
28:     add  $(\text{sig}(\text{TX}_{i,3,k_1,k_2}, pk_{p,i,k_1}))_{p \in [n] \setminus \{i\}}$  to sigs
29:     add  $(\text{sig}(\text{TX}_{i,3,k_1,k_2}, pk_{p,i,k_2}))_{p \in [n] \setminus \{i\}}$  to sigs
30:     add  $\text{SIGN}(\text{TX}_{i,3,k_1,k_2}, sk_{i,i,k_1}, \text{ANYPREVOUT})$  to sigs
31:     add  $\text{SIGN}(\text{TX}_{i,3,k_1,k_2}, sk_{i,i,k_2}, \text{ANYPREVOUT})$  to sigs
32:     input (SUBMIT,  $\text{TX}_{i,3,k_1,k_2}$ , sigs) to  $\mathcal{G}_{\text{Ledger}}$ 
33:   else if  $k_1 > 0$  then // only left neighbour has published its TX
34:     add  $(\text{sig}(\text{TX}_{i,2,k_1}, pk_{p,i,k_1}))_{p \in [n] \setminus \{i\}}$  to sigs
35:     add  $\text{SIGN}(\text{TX}_{i,2,k_1}, sk_{i,i,k_1}, \text{ANYPREVOUT})$  to sigs
36:     add  $\text{SIGN}(\text{TX}_{i,2,k_1}, sk_{i,\text{left},\text{fund},\text{old}}, \text{ANYPREVOUT})$  to sigs
37:     input (SUBMIT,  $\text{TX}_{i,2,k_1}$ , sigs) to  $\mathcal{G}_{\text{Ledger}}$ 
38:   else if  $k_2 > 0$  then // only right neighbour has published its TX
39:     add  $(\text{sig}(\text{TX}_{i,2,k_2}, pk_{p,i,k_2}))_{p \in [n] \setminus \{i\}}$  to sigs
40:     add  $\text{SIGN}(\text{TX}_{i,2,k_2}, sk_{i,i,k_2}, \text{ANYPREVOUT})$  to sigs
41:     add  $\text{SIGN}(\text{TX}_{i,2,k_2}, sk_{i,\text{right},\text{fund},\text{old}}, \text{ANYPREVOUT})$  to sigs
42:     input (SUBMIT,  $\text{TX}_{i,2,k_2}$ , sigs) to  $\mathcal{G}_{\text{Ledger}}$ 
43:   end if

```

Fig. 39.

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

```

1: // At most one of funder, fundee is defined
2: if  $\text{State} = \text{CLOSED}$  then output (CLOSED) to  $R$ 
3: if  $\text{State} = \text{GUEST PUNISHED}$  then output (GUEST PUNISHED) to  $R$ 
4: ensure  $\text{State} = \text{OPEN}$ 
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 IF 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: if  $i \in \{1, n\} \wedge (\text{TX}_{(i-1)+\frac{2}{n-1}(n-i),1} \in \Sigma \vee \exists k \in [n] : \text{TX}_{(i-1)+\frac{2}{n-1}(n-i),2,k} \in \Sigma)$ 
    then // we are an endpoint and our counterparty has closed – 1st subscript of
    TX is 2 if  $i = 1$  and  $n - 1$  if  $i = n$ 
12:   ignore all messages except for (CHECK IF CLOSING) by  $R$ 
13:    $\text{State} \leftarrow \text{CLOSING}$ 
14:   give up execution token // control goes to  $\mathcal{E}$ 
15: end if
16: let  $\text{tx}$  be the unique TX among  $\text{TX}_{i,1}, (\text{TX}_{i,2,k})_{k \in [n]}, (\text{TX}_{i,3,k_1,k_2})_{k_1,k_2 \in [n]}$ 
    that can be appended to  $\Sigma$  in a valid way // ignore invalid subscript
    combinations
17: let  $\text{sigs}$  be the set of stored signatures that sign  $\text{tx}$ 
18: add  $\text{SIGN}(\text{tx}, sk_{i,\text{left},\text{fund},\text{old}}, \text{ANYPREVOUT}), \text{SIGN}(\text{tx}, sk_{i,\text{right},\text{fund},\text{old}},$ 
     $\text{ANYPREVOUT}), (\text{SIGN}(\text{tx}, sk_{i,j,k}, \text{ANYPREVOUT}))_{j,k \in [n]}$  to  $\text{sigs}$  // ignore invalid
    signatures
19: ignore all messages except for (CHECK IF CLOSING) by  $R$ 
20:  $\text{State} \leftarrow \text{CLOSING}$ 
21: send (SUBMIT,  $\text{tx}, \text{sigs}$ ) to  $\mathcal{G}_{\text{Ledger}}$ 

```

Fig. 40.

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

```

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

```

```

11:   ensure that there exists an output with  $c_P + c_{\bar{P}} - c_{\text{virt}}$  coins and a
       $2/\{pk_{i,\text{fund,new}}, pk_{j,\text{fund,new}}\}$  spending method with expired/non-existent
      timelock and an output with  $c_{\text{virt}}$  coins and a  $pk_{i,\text{out}}$  spending method with
      expired/non-existent timelock in  $\Sigma$ 
12: end if
13:  $State \leftarrow \text{CLOSED}$ 
14: output (CLOSED) to  $R$ 

```

Fig. 41.

**Process VIRT – punishment handling**

```

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

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

12: On output (USED REVOCATION) by  $\text{host}_P$ :
13:   $State \leftarrow \text{GUEST PUNISHED}$ 
14:  if funder or fundee is defined then
15:    output (ENABLER USED REVOCATION) to it
16:  else // sibling is defined
17:    output (ENABLER USED REVOCATION) to sibling
18:  end if

```

Fig. 42.

**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  $(\text{OPEN}, c, \dots)$  by either  $\mathcal{E}$  or  $\bar{P}$ ,
- $P$  [has received  $(\text{FUND ME}, f_i, \dots)$  as input by another LN ITI while State was OPEN and subsequently  $P$  transitioned to OPEN State]  $n$  times,

- $P$  [has received  $(\text{PAY}, d_i)$  by  $\mathcal{E}$  while State was OPEN and  $P$  subsequently transitioned to OPEN State]  $m$  times,
- $P$  [has received  $(\text{GET PAID}, e_i)$  by  $\mathcal{E}$  while State was OPEN and  $P$  subsequently transitioned to OPEN State]  $l$  times.

Let  $\phi = 1$  if  $P = \text{Alice}$ , or  $\phi = 0$  if  $P = \text{Bob}$ . If  $P$  receives  $(\text{CLOSE})$  by  $\mathcal{E}$  and, if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$  the output of  $\text{host}_P$  is  $(\text{CLOSED})$ , then eventually the state obtained when  $P$  inputs  $(\text{READ})$  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.* We first note that, as signature forgeries only happen with negligible probability and only a polynomial number of signatures are verified by honest parties throughout an execution, the event in which any forged signature passes the verification of an honest party or of  $\mathcal{G}_{\text{Ledger}}$  happens only with negligible probability. We can therefore ignore this event throughout this proof and simply add a computationally negligible distance between  $\mathcal{E}$ 's outputs in the real and the ideal world at the end.

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

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

Let party  $P$  with a particular history. We will inductively prove that  $P$  satisfies the Lemma. The base case is when a channel is opened with **hops**  $= \mathcal{G}_{\text{Ledger}}$  and is closed right away, therefore  $H = ([], [g])$ , where  $g = c$  if  $P = \text{Alice}$

and  $g = 0$  if  $P = \text{Bob}$ .  $P$  can transition to the OPEN State for the first time only if all of the following have taken place:

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

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

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

- If  $P$  receives (FUND ME,  $f$ , ...) by a (local, trusted) LN ITI  $R$ , subsequently transitions back to the OPEN state (therefore moving to history  $(F, C')$  where  $C'$  is  $C + [-f]$ ) and finally receives (CLOSE) by  $\mathcal{E}$  and (CLOSED) by  $\text{host}_P$  before any further change to its history, then *eventually*  $P$ 's  $\mathcal{G}_{\text{Ledger}}$  state will contain  $h$   $(c_i, pk_{P,\text{out}})$  transaction outputs such that  $\sum_{i=1}^h c_i \geq \sum_{s \in C'} \sum_{x \in s} x$ .

Furthermore, given that  $P$  moves to the OPEN state after the (FUND ME, ...) message, it also sends (FUNDED, ...) to  $R$  (Fig. 17, l. 21). If subsequently the state of  $R$  transitions to OPEN (therefore obtaining history  $(F_R, C_R)$  where  $F_R = F + C$  and  $C_R = [[f]]$ ), and finally receives (CLOSE) by  $\mathcal{E}$  and (CLOSED)

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

By the induction hypothesis, before the funding procedure started  $P$  could close the channel and end up with a  $(c_P, pk_{P,\text{out}})$  output on-chain. When  $P$  is in the OPEN state and receives (FUND ME,  $f$ , ...), it can only move again to the OPEN state after doing the following state transitions: OPEN  $\rightarrow$  VIRTUALISING  $\rightarrow$  WAITING FOR REVOCATION  $\rightarrow$  WAITING FOR INBOUND REVOCATION  $\rightarrow$  WAITING FOR HOSTS READY  $\rightarrow$  OPEN. During this sequence of events, a new  $\text{host}_P$  is defined (Fig. 17, l. 6), new commitment transactions are negotiated with  $\bar{P}$  (Fig. 17, l. 9), control of the old funding output is handed over to  $\text{host}_P$  (Fig. 17, l. 11),  $\text{host}_P$  negotiates with its counterparty a new set of transactions and signatures that spend the aforementioned funding output and make available a new funding output with the keys  $pk'_{P,F}, pk'_{\bar{P},F}$  as  $P$  instructed (Fig. 35 and 37)

and the previous valid commitment transactions of both  $P$  and  $\bar{P}$  are invalidated (Fig. 9, l. 1 and l. 14 respectively). We note that the use of the `ANYPREVOUT` flag in all signatures that correspond to transaction inputs that may spend various different transaction outputs ensures that this is possible, as it avoids tying each input to a specific, predefined output. When  $P$  receives `(CLOSE)` by  $\mathcal{E}$ , it inputs `(CLOSE)` to `hostP` (Fig. 26, l. 4). As per the Lemma conditions, `hostP` will output `(CLOSED)`. This can only happen only when  $\mathcal{G}_{\text{Ledger}}$  contains a suitable output for both  $P$ 's and  $R$ 's channel (Fig. 41, and 4 ll. 5 respectively).

If the `host` of `hostP` is  $\mathcal{G}_{\text{Ledger}}$ , then the funding output  $o_{1,2} = (c_P + c_{\bar{P}}, 2/\{pk_{P,F}, pk_{\bar{P},F}\})$  for the  $A_1, A_2$  channel is already on-chain. Regarding the case in which `hostP`  $\neq \mathcal{G}_{\text{Ledger}}$ , after the funding procedure is complete, the new `hostP` will have as its `host` the old `hostP` of  $P$ . If the `(CLOSE)` sequence is initiated, the new `hostP` will follow the same steps that will be described below once the old `hostP` succeeds in closing the lower layer (Fig. 40, l. 5). The old `hostP` however will see no difference in its interface compared to what would happen if  $P$  had received `(CLOSE)` before the funding procedure, therefore it will successfully close by the induction hypothesis. Thereafter the process is identical to the one when the old `hostP` =  $\mathcal{G}_{\text{Ledger}}$ .

Moving on, `hostP` is either able to publish its  $\text{TX}_{1,1}$  (it has necessarily received a valid signature  $\text{sig}(\text{TX}_{1,1}, pk_{\bar{P},F})$  (Fig. 37, l. 39) by its counterparty before it moved to the `OPEN` state for the first time), or the output  $(c_P + c_{\bar{P}}, 2/\{pk_{P,F}, pk_{\bar{P},F}\})$  needed to spend  $\text{TX}_{1,1}$  has already been spent. The only other transactions that can spend it are  $\text{TX}_{2,1}$  and any of  $(\text{TX}_{2,2,k})_{k>2}$ , since these are the only transactions that spend the aforementioned output and that `hostP` has signed with  $sk_{P,F}$  (Fig. 37, ll. 33-37). The output can be also spent by old, revoked commitment transactions, but in that case `hostP` would not have output `(CLOSED)`;  $P$  would have instead detected this triggered by a `(CHECK CHAIN FOR CLOSED)` message by  $\mathcal{E}$  (Fig. 24) and would have moved to the `CLOSED` state on its own accord (lack of such a message by  $\mathcal{E}$  would lead  $P$  to become **negligent**, something that cannot happen according to the Lemma conditions). Every transaction among  $\text{TX}_{1,1}$ ,  $\text{TX}_{2,1}$ ,  $(\text{TX}_{2,2,k})_{k>2}$  has a  $(c_P + c_{\bar{P}} - f, 2/\{pk'_{P,F}, pk'_{\bar{P},F}\})$  output (Fig. 31, l. 18 and Fig. 30, ll. 27 and 91) which will end

up in  $\mathcal{G}_{\text{Ledger}}$  – call this output  $o_P$ . We will prove that at most  $\sum_{i=2}^{n-1} (t_i + p + s - 1)$

blocks after `(CLOSE)` is received by  $P$ , an output  $o_R$  with  $c_{\text{virt}}$  coins and a  $2/\{pk_{R,F}, pk_{\bar{R},F}\}$  spending condition without or with an expired timelock will be included in  $\mathcal{G}_{\text{Ledger}}$ . In case party  $A_2$  is idle, then  $o_{1,2}$  is consumed by  $\text{TX}_{1,1}$  and the timelock on its virtual output expires, therefore the required output  $o_R$  is on-chain. In case  $A_2$  is active, exactly one of  $\text{TX}_{2,1}$ ,  $(\text{TX}_{2,2,k})_{k>2}$  or  $(\text{TX}_{2,3,1,k})_{k>2}$  is a descendant of  $o_{1,2}$ ; if the transaction belongs to one of the two last transaction groups then necessarily  $\text{TX}_{1,1}$  is on-chain in some block height  $h$  and given the timelock on the virtual output of  $\text{TX}_{1,1}$ ,  $A_2$ 's transaction can be at most at block height  $h + t_2 + p + s - 1$ . If  $n = 3$  or  $k = n - 1$ , then  $A_2$ 's unique transaction has the required output  $o_R$  (without a timelock). The rest of the cases are covered by the following sequence of events:

Closing sequence

```

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

```

Fig. 43.

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

We will now prove that the newly funded party  $R$  can close its channel securely. After  $R$  receives (FUNDED,  $\text{host}_P, \dots$ ) by  $P$  and before moving to the OPEN state, it receives  $\text{sig}_{\bar{R},C,0}$  which is a valid signature on  $C_{R,0}$  by  $sk_{\bar{R},F}$  and sends  $\text{sig}_{R,C,0}$  which is a valid signature on  $C_{\bar{R},0}$  by  $sk_{R,F}$ . Both these transactions spend  $o_R$ . As we showed before, if  $R$  receives (CLOSE) by  $\mathcal{E}$  then  $o_R$  eventually ends up on-chain. After receiving (CLOSED) from  $\text{host}_P$ ,  $R$  attempts to add  $C_{R,0}$  to  $\mathcal{G}_{\text{Ledger}}$ , which may only fail if  $C_{\bar{R},0}$  ends up on-chain instead. Similar to the case of  $P$ , both these transactions have an  $(f, pk_{R,\text{out}})$  output.

This output of  $C_{R,0}$  is timelocked, but the alternative spending method cannot be used as  $R$  never signed a transaction that uses it (as it is reserved for revocation, which has not taken place yet) so the timelock will expire and the desired spending method will be available. We have now proven that if  $R$ 's channel is funded to completion then it can close its channel for a  $(f, pk_{R,\text{out}})$  output. We have therefore proven the first bullet.  $\square$

**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:*

- $\text{State}_P \neq \text{IGNORED}$ ,
- $P$  has transitioned to the OPEN State at least once. Additionally, if  $P = \text{Alice}$ , it has received  $(\text{OPEN}, c, \dots)$  by  $\mathcal{E}$  prior to transitioning to the OPEN State,
- $P$  [has received  $(\text{FUND ME}, f_i, \dots)$  as input by another  $\mathcal{F}_{\text{Chan}}/\text{LN ITI}$  while  $\text{State}_P = \text{OPEN}$  and  $P$  subsequently transitioned to OPEN State]  $n \geq 0$  times,
- $P$  [has received  $(\text{PAY}, d_i)$  by  $\mathcal{E}$  while  $\text{State}_P = \text{OPEN}$  and  $P$  subsequently transitioned to OPEN State]  $m \geq 0$  times,
- $P$  [has received  $(\text{GET PAID}, e_i)$  by  $\mathcal{E}$  while  $\text{State}_P = \text{OPEN}$  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  $(\text{CLOSE}, P)$  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  $\text{balance}_P$  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  $(\text{OPEN}, c, \dots)$  by  $\mathcal{E}$ , it stores  $c$  (Fig. 2, l. 10). If later  $\text{State}_P$  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  $\text{State}_P$  to OPEN (Fig. 2).
- Every time  $P$  receives input  $(\text{FUND ME}, f_i, \dots)$  by another party while  $\text{State}_P = \text{OPEN}$ ,  $P$  stores  $f_i$  (Fig. 4, l. 1). The next time  $\text{State}_P$  transitions to OPEN (if such a transition happens),  $\text{balance}_P$  is decremented by  $f_i$  (Fig. 4, l. 22). Therefore, if this cycle happens  $n \geq 0$  times,  $\text{balance}_P$  will be decremented by  $\sum_{i=1}^n f_i$  in total.
- Every time  $P$  receives input  $(\text{PAY}, d_i)$  by  $\mathcal{E}$  while  $\text{State}_P = \text{OPEN}$ ,  $d_i$  is stored (Fig. 3, l. 2). The next time  $\text{State}_P$  transitions to OPEN (if such a transition happens),  $\text{balance}_P$  is decremented by  $d_i$  (Fig. 3, l. 13). Therefore, if this cycle happens  $m \geq 0$  times,  $\text{balance}_P$  will be decremented by  $\sum_{i=1}^m d_i$  in total.



- Every time  $P$  receives input (GET PAID,  $e_i$ ) by  $\mathcal{E}$  while  $State_P = \text{OPEN}$ ,  $e_i$  is stored (Fig. 3, l. 7). The next time  $State_P$  transitions to OPEN (if such a transition happens)  $\text{balance}_P$  is incremented by  $e_i$  (Fig. 3, l. 19). Therefore, if this cycle happens  $l \geq 0$  times,  $\text{balance}_P$  will be incremented by  $\sum_{i=1}^l e_i$  in total.

On aggregate, after the above are completed and then  $\mathcal{F}_{\text{Chan}}$  receives (CLOSE,  $P$ ) by  $\mathcal{S}$ , it is  $\text{balance}_P = c - \sum_{i=1}^n f_i - \sum_{i=1}^m d_i + \sum_{i=1}^l e_i$  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  $\text{host}_P$  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  $\text{host}_P$  is GUEST PUNISHED (Fig. 42, ll. 1 or 12), so in case  $P$  receives (CLOSE) by  $\mathcal{E}$  the output of  $\text{host}_P$  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  $State_P = \text{UNINIT}$ , 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

$R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$ ,  $\text{State}_P$  transitions to PENDING FUND, subsequently when a command to define a new VIRT ITI through  $P$  is intercepted by  $\mathcal{F}_{\text{Chan}}$ ,  $\text{State}_P$  transitions to TENTATIVE FUND and afterwards when  $\mathcal{S}$  sends (FUND) to  $\mathcal{F}_{\text{Chan}}$ ,  $\text{State}_P$  transitions to SYNC FUND. In parallel, if  $\text{State}_{\bar{P}} = \text{IGNORED}$ , then  $\text{State}_P$  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  $\text{State}_P$  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  $\mathcal{E}$ ,  $\text{State}_P$  transitions to TENTATIVE PAY and subsequently when  $\mathcal{S}$  sends (PAY) to  $\mathcal{F}_{\text{Chan}}$ ,  $\text{State}_P$  transitions to (SYNC PAY,  $d$ ). In parallel, if  $\text{State}_{\bar{P}} = \text{IGNORED}$ , then  $\text{State}_P$  transitions directly back to OPEN. If on the other hand  $\text{State}_{\bar{P}} = \text{OPEN}$  and  $\mathcal{F}_{\text{Chan}}$  receives (GET PAID,  $d$ ) by  $\mathcal{E}$  addressed to  $\bar{P}$ ,  $\text{State}_{\bar{P}}$  transitions to TENTATIVE GET PAID. On receiving the aforementioned (PAY) message by  $\mathcal{S}$  and given that  $\text{State}_{\bar{P}} = \text{TENTATIVE GET PAID}$ ,  $\mathcal{F}_{\text{Chan}}$  also sets  $\text{State}_{\bar{P}}$  to SYNC GET PAID. Then both  $\text{State}_P$  and  $\text{State}_{\bar{P}}$  transition simultaneously to OPEN (Fig. 3). This sequence of events may repeat any  $m \geq 0$  times. We observe that throughout these steps, honest simulated  $P$  has received (PAY,  $d$ ) and that  $\mathcal{S}$  only sends (PAY) when all honest simulated parties have completed sending or receiving the payment (Fig. 7, l. 16), so the fourth condition of Lemma 1 holds with the same  $m$  as that of Lemma 2. As far as the fifth condition of Lemma 2 goes, we observe that this case is symmetric to the one discussed for its fourth condition above if we swap  $P$  and  $\bar{P}$ , therefore we deduce that if Lemma 2 holds with some  $l$ , then Lemma 1 holds with the same  $l$ .

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

Now, given that  $\mathcal{S}$  internally simulates faithfully both LN parties and that  $\mathcal{F}_{\text{Chan}}$  relinquishes to  $\mathcal{S}$  complete control of the external communication of the parties as long as it does not halt, we deduce that  $\mathcal{S}$  replicates the behaviour of the aforementioned real world. By combining these facts with the consequences of the two Lemmas and the check that leads  $\mathcal{F}_{\text{Chan}}$  to halt if it fails (Fig. 5, l. 6), we deduce that if the conditions of Lemma 2 hold, then the functionality halts only with negligible probability.

In the second proof step, we show that if the conditions of Lemma 2 do not hold, then the check of Fig. 5, l. 6 never takes place. We first discuss the  $\text{State}_P = \text{IGNORED}$  case. We observe that the  $\text{IGNORED State}$  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  $(\text{CLOSED}, P)$  by  $\mathcal{S}$  when  $\text{State}_P \neq \text{IGNORED}$  (Fig. 5, l. 3). We deduce that, once  $\text{State}_P = \text{IGNORED}$ , the balance check will not happen. Moving to the case where  $\text{State}_P$  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  $P = \text{Alice}$ , it is impossible to reach the OPEN state without receiving input  $(\text{OPEN}, c, \dots)$  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,*

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

*Proof.* By inspection of Figs. 1 and 6 we can deduce that for a particular  $\mathcal{E}$ , in the ideal world execution  $\text{EXEC}_{\mathcal{S}_{\mathcal{A}}, \mathcal{E}}^{\mathcal{F}_{\text{Chan}}, \mathcal{G}_{\text{Ledger}}}$ ,  $\mathcal{S}_{\mathcal{A}}$  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  $\mathcal{F}_{\text{Chan}}$  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

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