

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

Functionality $\mathcal{F}_{\text{Chan}}$ – relay rules

- Every input by \mathcal{E} is first handled according to the corresponding rule in Fig 2 or Fig 3 (if any) and subsequently is sent to \mathcal{A} .
- Every message $(\text{msg}, P, R, \text{mode})$ by \mathcal{A} ($\text{mode} \in \{\text{input}, \text{output}, \text{message}\}$, $P \in \{\text{Alice}, \text{Bob}\}$) is first handled according to the corresponding rule in Fig 2 or Fig 3 (if any). Subsequently msg is relayed to R as P by means of mode . If R and/or mode are undefined, no relaying takes place.
- If an “ensure” fails, the rest of the code is not run but the relaying happens as usual.

Fig. 1.

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

```

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

6: On (CORRUPT,  $P$ ) by  $\mathcal{A}$ :
7:    $\text{is\_corrupted}_P \leftarrow \text{True}$ 

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

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

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

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

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

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

```

Fig. 2.

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

```

1: On (PAY FAIL, __, __, __) by  $\mathcal{A}$ :
2:   ensure  $State = \text{PAYING}$ 
3:   forget  $c, S, \bar{S}$ 
4:    $State \leftarrow \text{OPEN}$ 

5: On (FUND,  $x$ ) to  $P$  by  $R$ :
6:   ensure  $R$  is an  $\mathcal{F}_{\text{Chan}}$  party
7:   ensure  $State = \text{OPEN}$ 
8:    $c \leftarrow x$ 
9:   if  $P = \text{Alice}$  then  $S \leftarrow \text{Alice}$  else  $S \leftarrow \text{Bob}$ 
10:   $State \leftarrow \text{FUNDING}$ 

11: On (FUND OK, __, __, __) by  $\mathcal{A}$ :
12:   ensure  $State = \text{FUNDING}$ 
13:    $\text{balance}_S \leftarrow \text{balance}_S - c$ 
14:   forget  $c, S$ 
15:    $State \leftarrow \text{OPEN}$ 

16: On (FUND FAIL, __, __, __) by  $\mathcal{A}$ :
17:   ensure  $State = \text{FUNDING}$ 
18:   forget  $c, S$ 
19:    $State \leftarrow \text{OPEN}$ 

20: On (CLOSE) by  $\mathcal{E}$ :
21:   ensure  $State = \text{OPEN}$ 
22:    $State \leftarrow \text{CLOSING}$ 

23: On (CLOSE, __, __, __) by  $\mathcal{A}$ :
24:   ensure  $State \in \{\text{OPEN}, \text{CLOSING}\}$ 
25:   for  $P \in \{A, B\}$  do
26:     if  $\text{is\_corrupted}_P = \text{False}$  then
27:       input (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $P$  and assign output to  $\Sigma_P$ 
28:        $\text{coins}_P \leftarrow$  sum of coins exclusively spendable by  $pk_P$  in  $\Sigma_P$ 
29:     end if
30:   end for
31:   if  $(\text{is\_corrupted}_A \vee \text{coins}_A \geq \text{balance}_A)$ 
 $\wedge (\text{is\_corrupted}_B \vee \text{coins}_B \geq \text{balance}_B)$  then
32:      $State \leftarrow \text{CLOSED}$ 
33:   else // balance security is broken
34:     halt
35:   end if

```

Fig. 3.

Process LN – init

```

1: // When not specified, input comes from and output goes to  $\mathcal{E}$  in the real or
    $P \in \{Alice, Bob\}$  in the ideal world.
2: // In the real world, the player knows whether it is Alice (funder) or Bob
   (fundee). The activated party is  $P$  and the counterparty is  $\bar{P}$ .
3: On (INIT,  $pk_{P,out}$ ):
4:   ensure  $State^P = \perp$ 
5:    $State^P \leftarrow \text{INIT}$ 
6:   store  $pk_{P,out}$ 
7:    $(c_A, c_B, \text{locked}_A, \text{locked}_B) \leftarrow (0, 0, 0, 0)$ 
8:    $(\text{paid\_out}, \text{paid\_in}) \leftarrow (\emptyset, \emptyset)$ 
9:   output (INIT OK)

10: On (TOP UP):
11:   ensure  $P = Alice$  // activated party is the funder
12:   ensure  $State^P = \text{INIT}$ 
13:    $(sk_{P,chain}, pk_{P,chain}) \leftarrow \text{KEYGEN}()$ 
14:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $P$  and assign output to  $\Sigma$ 
15:   output (TOP UP TO,  $pk_{P,chain}$ )
16:   while  $\nexists tx \in \Sigma, c_{P,chain} : (c_{P,chain}, pk_{P,chain}) \in tx.outputs$  do
17:     // while waiting, all other messages by  $P$  are ignored
18:     wait for input (CHECK TOP UP)
19:     input (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $P$  and assign output to  $\Sigma$ 
20:   end while
21:    $State^P \leftarrow \text{TOPPED UP}$ 
22:   output (TOP UP OK,  $c_{P,chain}$ )

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

```

Fig. 4.

Process LN – methods used by VIRT

```

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

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

```

Fig. 5.

Process LN.EXCHANGEOPENKEYS()

```

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

```

Fig. 6.

Process LN.PREPAREBASE()

```

1: if hops =  $\mathcal{G}_{\text{Ledger}}$  then // opening base channel
2:    $F \leftarrow \text{TX}$  {input:  $(c, pk_{A,\text{chain}})$ , output:  $(c, 3 \wedge 2 / \{pk_{A,F}, pk_{B,F}\})$ }
3:    $\text{host}_A \leftarrow \mathcal{G}_{\text{Ledger}}$ 
4:   if ideal world then
5:      $\text{host}_B \leftarrow \mathcal{G}_{\text{Ledger}}$ 
6:      $\text{State}^B \leftarrow \text{WAITING TO CHECK FUNDING}$ 
7:   end if
8: else // opening virtual channel
9:   if ideal world then
10:    input (FUND ME,  $\bar{P}$ , hops,  $c$ ,  $pk_{A,F}$ ,  $pk_{B,F}$ ) to hops[0].left and expect
    output (FUNDED,  $\text{host}_{\bar{P}}$ ) to fundee by hops[-1].right // ignore any other
    message
11:    input (FUND ACK) as fundee to hops[-1].right and expect output
    (FUNDED,  $\text{host}_P$ ) to funder by hops[0].left
12:   else // real world
13:    input (FUND ME, Alice, Bob, hops,  $c$ ,  $pk_{A,F}$ ,  $pk_{B,F}$ ) to hops[0].left and
    expect output (FUNDED,  $\text{host}_P$ ) // ignore any other message
14:   end if
15: end if

```

Fig. 7.

Process LN.EXCHANGEOPENSIGS()

```

1:  $C_{A,0} \leftarrow \text{TX}$  {input:  $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$ , outputs:  $(c, (pk_{A,\text{out}} + t) \vee$ 
    $2/\{pk_{A,R}, pk_{B,R}\})$ ,  $(0, pk_{B,\text{out}})\}$ 
2:  $C_{B,0} \leftarrow \text{TX}$  {input:  $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$ , outputs:  $(c, pk_{A,\text{out}})$ ,  $(0,$ 
    $(pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})\}$ 
3:  $\text{sig}_{A,C,0} \leftarrow \text{SIGN}(C_{B,0}, sk_{A,F})$ 
4:  $\text{State}^A \leftarrow \text{WAITING FOR COMM SIG}$ 
5: send (FUNDING CREATED,  $(c, pk_{A,\text{chain}})$ ,  $\text{sig}_{A,C,0}$ ) to fundee
6: ensure  $\text{State}^B = \text{WAITING FOR COMM SIG}$  // if opening virtual channel, we
   have received (FUNDED, host_fundee) by hops[-1].right (c.f. Fig 10, line ??)
7: if hops =  $\mathcal{G}_{\text{Ledger}}$  then // opening base channel
8:    $F \leftarrow \text{TX}$  {input:  $(c, pk_{A,\text{chain}})$ , output:  $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})\}$ 
9: end if
10:  $C_{B,0} \leftarrow \text{TX}$  {input:  $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$ , outputs:  $(c, pk_{A,\text{out}})$ ,  $(0,$ 
    $(pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})\}$ 
11: ensure  $\text{VERIFY}(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F}) = \text{True}$ 
12:  $C_{A,0} \leftarrow \text{TX}$  {input:  $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$ , outputs:  $(c, (pk_{A,\text{out}} + t) \vee$ 
    $2/\{pk_{A,R}, pk_{B,R}\})$ ,  $(0, pk_{B,\text{out}})\}$ 
13:  $\text{sig}_{B,C,0} \leftarrow \text{SIGN}(C_{A,0}, sk_{B,F})$ 
14: if hops =  $\mathcal{G}_{\text{Ledger}}$  then // opening base channel
15:    $\text{State}^B \leftarrow \text{WAITING TO CHECK FUNDING}$ 
16: else // opening virtual channel
17:    $c_A \leftarrow c$ ;  $c_B \leftarrow 0$ ;  $i \leftarrow 0$ 
18:    $\text{State}^B \leftarrow \text{OPEN}$ 
19: end if
20: reply (FUNDING SIGNED,  $\text{sig}_{B,C,0}$ )
21: ensure  $\text{State}^A = \text{WAITING FOR COMM SIG}$ 
22: ensure  $\text{VERIFY}(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F}) = \text{True}$ 

```

Fig. 8.

Process LN.COMMITBASE()

```

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

```

Fig. 9.

Process LN – external open messages for *Bob*

```

1: On input (CHECK FUNDING): // real or ideal world
2:   ensure  $State^B = \text{WAITING TO CHECK FUNDING}$ 
3:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
4:   if  $F \in \Sigma$  then
5:      $State^B \leftarrow \text{OPEN}$ 
6:     reply (OPEN OK)
7:   end if

8: On output (FUNDED,  $\text{host}_P$ ) by  $\text{hops}[-1].\text{right}$ : // real world
9:   ensure  $State^B = \text{WAITING FOR FUNDED}$ 
10:  store  $\text{host}_P$  // we will talk directly to  $\text{host}_P$ 
11:   $State^B \leftarrow \text{WAITING FOR COMM SIG}$ 
12:  reply (FUND ACK)

13: On output (CHECK KEYS,  $(pk_1, pk_2)$ ) by  $\text{hops}[-1].\text{right}$ :
14:  ensure  $State^B = \text{WAITING FOR CHECK KEYS}$ 
15:  ensure  $pk_1 = pk_{A,F} \wedge pk_2 = pk_{B,F}$ 
16:   $State^B \leftarrow \text{WAITING FOR FUNDED}$ 
17:  reply (KEYS OK)

```

Fig. 10.

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

```

1: // fundee is Bob
2: ensure  $P = Alice$  // activated party is the funder
3: if hops =  $\mathcal{G}_{\text{Ledger}}$  then // opening base channel
4:   ensure  $State^A = \text{TOPPED UP}$ 
5:   ensure  $c = c_{A, \text{chain}}$ 
6: else // opening virtual channel
7:   ensure  $\text{len}(\text{hops}) \geq 2$  // cannot open a virtual over 1 channel
8: end if
9: LN.EXCHANGEOPENKEYS()
10: LN.PREPAREBASE()
11: if real world then
12:   LN.EXCHANGEOPENSIGS()
13: else // ideal world
14:   send (FUNDING CREATED, (c,  $pk_{A, \text{chain}}$ )) to  $\mathcal{A}$  and expect reply (FUNDING
      CREATED OK)
15: end if
16: if hops =  $\mathcal{G}_{\text{Ledger}}$  then
17:   LN.COMMITBASE()
18: else if ideal world then // opening virtual channel in ideal world
19:    $State^B \leftarrow \text{OPEN}$ 
20: end if
21:  $c_A \leftarrow c$ ;  $c_B \leftarrow 0$ ;  $i \leftarrow 0$ 
22:  $State^A \leftarrow \text{OPEN}$ 
23: output (OPEN OK, c, fundee, hops)

```

Fig. 11.

Process LN.UPDATEFORVIRTUAL()

```

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

```

Fig. 12.

Process LN – virtualise start and end

```

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

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

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

```

Fig. 13.

Process LN – virtualise hops

```

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

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

```

Fig. 14.

Process LN.SIGNATURESROUNDTrip()

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

Fig. 15.

Process LN.REVOCATIONSTRIP()

```

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

```

Fig. 16.

Process LN – On (PAY, x):

```

1: ensure  $State^P = \text{OPEN} \wedge c_P \geq x$ 
2: if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge P$  has a host_sibling then // we are intermediary
   channel
3:   ensure  $c_{\text{sib,rem}} \geq c_P - x \wedge c_{\bar{P}} + x \geq c_{\text{sib}}$  // avoid value loss by grieving
   attack: one counterparty closes with old version, the other stays idle forever
4: end if
5: if ideal world then
6:   send (PAY,  $x$ ) to  $\mathcal{A}$  and expect reply (PAY OK)
7:   ensure  $State^{\bar{P}} = \text{OPEN}$ 
8:    $c_P \leftarrow c_P - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x; i \leftarrow i + 1$ 
9:   if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge P$  has 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: else
18:   LN.SIGNATURESROUNDTrip()
19:   LN.REVOCATIONSTrip()
20: end if
21: // No output is given to the caller, this is intentional

```

Fig. 17.

Process LN – On (CHECK FOR LATERAL CLOSE):

```

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

```

Fig. 18.

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

```

1: ensure  $State^P \notin \{\perp, \text{INIT}, \text{TOPPED UP}\}$  // channel open
2: // even virtual channels check  $\mathcal{G}_{\text{Ledger}}$  directly. This is intentional
3: send (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $P$  and assign reply to  $\Sigma$ 
4: if  $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$  then // counterparty has closed maliciously
5:    $State^P \leftarrow \text{CLOSING}$ 
6:   LN.SUBMITANDCHECKREVOCATION( $j$ )
7:    $State^P \leftarrow \text{CLOSED}$ 
8: end if

```

Fig. 19.

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  $\mathcal{G}_{\text{Ledger}}$  as  $P$ 
3: while  $\nexists R_{P,j} \in \Sigma$  do
4:   wait for input (CHECK REVOCATION) // ignore other messages
5:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $P$  and assign output to  $\Sigma$ 
6: end while

```

Fig. 20.

Process LN – On (CLOSE):

```

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

```

Fig. 21.

TODO: explicitly ensure that all hosts share the same sid – only if we decide that a functionality is needed after all

Process VIRT

```

1: On input (INIT,  $\text{host}_P$ ,  $\bar{P}$ , sibling, fundee,  $(sk_{\text{loc,virt}}, pk_{\text{loc,virt}})$ ,  $pk_{\text{rem,virt}}$ ,
    $pk_{\text{sib,rem,virt}}$ ,  $(sk_{\text{loc,F}}, pk_{\text{loc,F}})$ ,  $pk_{\text{rem,F}}$ ,  $c_{\text{guest}}$ ) by guest:
2:   store message contents and guest // sibling,  $pk_{\text{sib},\bar{P},F}$  are missing for
   edge nodes, fundee is present only in last node
3:   output (HOST INIT OK) to guest

4: On input (HOST ME, funder, fundee,  $\bar{P}$ ,  $\text{host}_P$ ,  $c_{\text{guest}}$ ,  $pk_{\text{left,guest}}$ ,  $pk_{\text{right,guest}}$ ,
    $(sk_{\text{loc,virt}}, pk_{\text{loc,virt}})$ ,  $(sk_{\text{loc,F}}, pk_{\text{loc,F}})$ ,  $pk_{\text{rem,F}}$ ,  $pk_{\text{rem,virt}}$ ) by guest:
5:   ensure VIRT.CIRCULATEKEYSANDCOINS() returns (OK)
6:   ensure VIRT.CIRCULATEVIRTUALSIGs() returns (OK)
7:   ensure VIRT.CIRCULATEFUNDINGSIGs() returns (OK)
8:   ensure VIRT.CIRCULATEREVOCATIONS() returns (OK)
9:   output (HOSTS READY) to guest

10: CIRCULATEKEYSANDCOINS(left_data):
11:   if left_data is given as argument then // we are not host_funder
12:     if we have a sibling then // we are not host_fundee
13:       input (KEYS AND COINS FORWARD, (left_data,  $(sk_{\text{loc,virt}}, pk_{\text{loc,virt}})$ ,
    $(sk_{\text{loc,F}}, pk_{\text{loc,F}})$ ,  $pk_{\text{rem,F}}$ ,  $c_P$ ,  $c_{\bar{P}}$ ) to sibling
14:       store input as left_data
15:       parse left_data as far_left_data,  $(sk_{\text{loc,virt}}, pk_{\text{loc,virt}})$ ,  $(sk_{\text{sib,F}}$ ,
    $pk_{\text{sib,rem,F}}$ ,  $pk_{\text{sib,rem,F}}$ ,  $c_{\text{sib}}$ ,  $c_{\text{sib,rem}}$  // remove parentheses as necessary
16:       call VIRT.CIRCULATEKEYSANDCOINS(left_data) of  $\bar{P}$  and assign
   returned value to right_data
17:       parse right_data as far_right_data,  $pk_{\text{rem,virt}}$ 
18:       output (KEYS AND COINS BACK, right_data,  $(sk_{\text{loc,F}}, pk_{\text{loc,F}})$ ,
    $pk_{\text{rem,F}}$ ,  $c_P$ ,  $c_{\bar{P}}$ )
19:       store output as right_data
20:       parse right_data as far_right_data,  $(sk_{\text{sib,F}}, pk_{\text{sib,F}})$ ,  $pk_{\text{sib,rem,F}}$ ,
    $c_{\text{sib}}$ ,  $c_{\text{sib,rem}}$ 
21:       return (right_data,  $pk_{\text{loc,virt}}$ )
22:     else // we are host_fundee
23:       extract  $(pk_{\text{left,guest}}, pk_{\text{right,guest}})$  from left_data
24:       output (CHECK KEYS,  $(pk_{\text{left,guest}}, pk_{\text{right,guest}})$ ) to fundee and
   expect reply (KEYS OK)
25:       return  $pk_{\text{loc,virt}}$ 
26:     end if
27:   else // we are host_funder
28:     call VIRT.CIRCULATEKEYSANDCOINS( $pk_{\text{loc,virt}}$ ,  $(pk_{\text{left,guest}}, pk_{\text{right,guest}})$ )
   of  $\bar{P}$  and assign returned value to right_data
29:     return (OK)
30:   end if

```

Fig. 22.

Process VIRT

```

1: GETMIDTXS( $c_{\text{guest}}, c_{\text{loc}}, c_{\text{rem}}, c_{\text{sib}}, c_{\text{sibRem}}, pk_{\text{left,fund}}, pk_{\text{loc,fund}}, pk_{\text{sib,fund}},$ 
 $pk_{\text{right,fund}}, pk_{\text{left,virt}}, pk_{\text{loc,virt}}, pk_{\text{sib,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}},$ 
 $pk_{\text{loc,out}}, \{pk_{\text{sec},i}\}_{i \in 1 \dots n}$ ):
2:   ensure  $c_{\text{sibRem}} \geq c_{\text{guest}} \wedge c_{\text{loc}} \geq c_{\text{guest}}$ 
3:    $c_{\text{left}} \leftarrow c_{\text{sib}} + c_{\text{sibRem}}; c_{\text{right}} \leftarrow c_{\text{loc}} + c_{\text{rem}}$ 
4:    $\text{left\_fund} \leftarrow 2 / \{pk_{\text{left,fund}}, pk_{\text{loc,fund}}\}$ 
5:    $\text{right\_fund} \leftarrow 2 / \{pk_{\text{sib,fund}}, pk_{\text{right,fund}}\}$ 
6:    $\text{left\_virt} \leftarrow 2 / \{pk_{\text{left,virt}}, pk_{\text{loc,virt}}\}$ 
7:    $\text{left\_virt\_checked} \leftarrow 4 / \{pk_{\text{left,virt}}, pk_{\text{loc,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$ 
8:    $\text{right\_virt} \leftarrow 2 / \{pk_{\text{sib,virt}}, pk_{\text{right,virt}}\}$ 
9:    $\text{right\_virt\_checked} \leftarrow 4 / \{pk_{\text{sib,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$ 
10:   $\text{left\_out\_checked} \leftarrow (2 \wedge \text{left\_virt\_checked}) \vee (3 \wedge \text{left\_virt} + t)$ 
11:   $\text{right\_out} \leftarrow (1 \wedge \text{right\_virt}) \vee (3 \wedge \text{right\_virt} + t)$ 
12:   $\text{right\_out\_checked} \leftarrow (1 \wedge \text{right\_virt\_checked}) \vee (3 \wedge \text{right\_virt} + t)$ 
13:   $\text{guest\_all} \leftarrow 5 \wedge n / \{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1 \dots n}\}\}$ 
14:   $\text{guest\_out} \leftarrow 4 \wedge 2 / \{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$ 
15:   $\text{guest} \leftarrow (\text{guest\_out} + t) \vee \text{guest\_all}$ 
16:   $\text{TX}_{\text{none}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}}, \text{left\_fund}), (c_{\text{right}}, \text{right\_fund})), \text{outputs:}$ 
 $((c_{\text{left}} - c_{\text{guest}}, \text{left\_out\_checked}), (c_{\text{right}} - c_{\text{guest}}, \text{right\_out\_checked}),$ 
 $(c_{\text{guest}}, pk_{\text{loc,out}}), (c_{\text{guest}}, \text{guest})) \}$ 
17:   $\text{TX}_{\text{left}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}} - c_{\text{guest}}, 1 \wedge \text{left\_virt\_checked}), (c_{\text{right}},$ 
 $\text{right\_fund})), \text{outputs: } ((c_{\text{left}} - c_{\text{guest}}, 3 \wedge \text{left\_virt}), (c_{\text{right}} - c_{\text{guest}},$ 
 $\text{right\_out\_checked}), (c_{\text{guest}}, pk_{\text{loc,out}})) \}$ 
18:   $\text{TX}_{\text{right}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}}, \text{left\_fund}), (c_{\text{right}} - c_{\text{guest}}, 2 \wedge$ 
 $\text{right\_virt\_checked}), (c_{\text{guest}}, \text{guest\_all})), \text{outputs: } ((c_{\text{left}} - c_{\text{guest}},$ 
 $\text{left\_out\_checked}), (c_{\text{right}} - c_{\text{guest}}, 3 \wedge \text{right\_virt}), (c_{\text{guest}}, pk_{\text{loc,out}}), (c_{\text{guest}},$ 
 $\text{guest})) \}$ 
19:   $\text{TX}_{\text{both}} \leftarrow \text{TX} \{ \text{inputs: } ((c_{\text{left}} - c_{\text{guest}}, 1 \wedge \text{left\_virt\_checked}),$ 
 $(c_{\text{right}} - c_{\text{guest}}, 2 \wedge \text{right\_virt\_checked}), (c_{\text{guest}}, \text{guest\_all})), \text{outputs:}$ 
 $((c_{\text{left}} - c_{\text{guest}}, 3 \wedge \text{left\_virt}), (c_{\text{right}} - c_{\text{guest}}, 3 \wedge \text{right\_virt}),$ 
 $(c_{\text{guest}}, pk_{\text{loc,out}})) \}$ 
20:  return  $(\text{TX}_{\text{none}}, \text{TX}_{\text{left}}, \text{TX}_{\text{right}}, \text{TX}_{\text{both}})$ 

```

Fig. 23.

Process VIRT

```

1: // left and right refer to the two counterparties, with left being the one closer
   to the funder. Note difference with left/right meaning in VIRT.GETMIDTXs.
2: GETEDGETXs( $c_{\text{guest}}, c_{\text{left}}, c_{\text{right}}, pk_{\text{left},\text{fund}}, pk_{\text{right},\text{fund}}, pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}},$ 
    $pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}, \{pk_{\text{sec},i}\}_{i \in 1 \dots n}, \text{is\_funder}$ ):
3:   ensure  $c_{\text{left}} \geq c_{\text{guest}}$ 
4:    $c_{\text{tot}} \leftarrow c_{\text{left}} + c_{\text{right}}$ 
5:    $\text{fund} \leftarrow 2/\{pk_{\text{left},\text{fund}}, pk_{\text{right},\text{fund}}\}$ 
6:    $\text{virt} \leftarrow 2/\{pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}}\}$ 
7:    $\text{virt\_checked} \leftarrow 4/\{pk_{\text{left},\text{virt}}, pk_{\text{right},\text{virt}}, pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}\}$ 
8:   if  $\text{is\_funder} = \text{True}$  then
9:      $\text{out} \leftarrow (1 \wedge \text{virt\_checked}) \vee (3 \wedge \text{virt} + t)$ 
10:  else // TXs belong to fundee
11:     $\text{out} \leftarrow (2 \wedge \text{virt\_checked}) \vee (3 \wedge \text{virt} + t)$ 
12:  end if
13:   $\text{guest\_all} \leftarrow 5 \wedge n/\{pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}, \{pk_{\text{sec},1 \dots n}\}\}$ 
14:   $\text{guest\_out} \leftarrow 4 \wedge 2/\{pk_{\text{left},\text{guest}}, pk_{\text{right},\text{guest}}\}$ 
15:   $\text{guest} \leftarrow (\text{guest\_out} + t) \vee \text{guest\_all}$ 
16:   $\text{TX}_{\text{base}} \leftarrow \text{TX} \{ \text{input: } (c_{\text{tot}}, \text{fund}), \text{outputs: } ((c_{\text{tot}} - c_{\text{guest}}, \text{out}), (c_{\text{guest}},$ 
     $\text{guest})) \}$ 
17:  return  $\text{TX}_{\text{base}}$ 

```

Fig. 24.

Process VIRT.SIBLINGSIGS()

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

Fig. 25.

Process VIRT.INTERMEDIARYSIGS()

```

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

```

Fig. 26.

Process VIRT.HOSTFUNDEESIGS()

```

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

```

Fig. 27.

Process VIRT.HOSTFUNDERSIGS()

```

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

```

Fig. 28.

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

```

1: if  $\text{sigs}_{\text{byLeft}}$  is given as argument then // we are not host_funder
2:   if we have a sibling then // we are not host_fundee
3:     return  $\text{VIRT.INTERMEDIARYSIGS}()$ 
4:   else // we are host_fundee
5:     return  $\text{VIRT.HOSTFUNDEESIGS}()$ 
6:   end if
7: else // we are host_funder
8:   return  $\text{VIRT.HOSTFUNDERSIGS}()$ 
9: end if

```

Fig. 29.

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

```

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

```

Fig. 30.

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:   call VIRT.CIRCULATEREVOCATIONS(revoc_for_next) of  $\bar{P}$  and assign
   returned value to revoc_by_next
6:   ensure guest.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK)
7:   return (OK)
8: end if
9: if we have a sibling then // we are not host_fundee nor host_funder
10:  input (VIRTUAL REVOCATION FORWARD) to sibling
11:  revoc_for_next ← guest.REVOKEPREVIOUS()
12:  call VIRT.CIRCULATEREVOCATIONS(revoc_for_next) of  $\bar{P}$  and assign
   output to revoc_by_next
13:  ensure guest.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK)
14:  output (VIRTUAL REVOCATION BACK)
15: end if
16: revoc_for_prev ← guest.REVOKEPREVIOUS()
17: output (HOSTS READY) to guest and expect reply (HOST ACK)
18: return revoc_for_prev // we are not host_fundee nor host_funder

```

Fig. 31.

Process VIRT – poll

```

1: On input (CHECK FOR LATERAL CLOSE) by  $R \in \{\text{guest, funder, fundee}\}$ :
2:   input (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign output to  $\Sigma$ 
3:    $\text{prev\_went\_on\_chain} \leftarrow \text{TX}_{\text{prev, left}} \in \Sigma \wedge \text{TX}_{\text{prev, both}} \in \Sigma$ 
4:    $\text{next\_went\_on\_chain} \leftarrow \text{TX}_{\text{next, right}} \in \Sigma \wedge \text{TX}_{\text{next, both}} \in \Sigma$ 
5:   if  $\text{prev\_went\_on\_chain} \vee \text{next\_went\_on\_chain}$  then
6:     ignore all messages except for (CHECK CHAIN FOR CLOSING) by  $R$ 
7:      $\text{State} \leftarrow \text{CLOSING}$ 
8:   end if
9:   if  $\text{prev\_went\_on\_chain} \wedge \text{next\_went\_on\_chain}$  then
10:    VIRT.SIGNANDSUBMIT( $\text{TX}_{\text{loc, both}}, \text{sigs}_{\text{loc, both}}$ )
11:   else if  $\text{prev\_went\_on\_chain}$  then
12:    VIRT.SIGNANDSUBMIT( $\text{TX}_{\text{loc, left}}, \text{sigs}_{\text{loc, left}}$ )
13:   else if  $\text{next\_went\_on\_chain}$  then
14:    VIRT.SIGNANDSUBMIT( $\text{TX}_{\text{loc, right}}, \text{sigs}_{\text{loc, right}}$ )
15:   end if

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

```

Fig. 32.

Process VIRT – close

```

1: On input (CLOSE) by  $R \in \{\text{guest}, \text{funder}, \text{funder}\}$ : // At most one of funder,
   fundee is defined
2:   if  $State = \text{CLOSED}$  then
3:     output (CLOSED) to  $R$ 
4:   end if
5:   ensure  $State = \text{OPEN}$ 
6:   if  $\text{host}_P \neq \mathcal{G}_{\text{Ledger}}$  then // host is a VIRT
7:     ignore all messages except for output (CLOSED) by host. Also relay to
     hostP any (CHECK CHAIN FOR CLOSING) input received
8:     input (CLOSE) to hostP
9:   end if
10:  // if we have a hostP, continue from here on output (CLOSED) by it
11:  send (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $R$  and assign reply to  $\Sigma$ 
12:  let tx be the unique valid TX for  $\Sigma$  among ( $\text{TX}_{\text{loc}, \text{none}}$ ,  $\text{TX}_{\text{loc}, \text{left}}$ ,
      $\text{TX}_{\text{loc}, \text{right}}$ ,  $\text{TX}_{\text{loc}, \text{both}}$ ) // if we are not an intermediary, only the first exists
13:  let sigs be the corresponding set of signatures among ( $\text{sigs}_{\text{loc}, \text{none}}$ ,
      $\text{sigs}_{\text{loc}, \text{left}}$ ,  $\text{sigs}_{\text{loc}, \text{right}}$ ,  $\text{sigs}_{\text{loc}, \text{both}}$ )
14:  add  $\text{SIGN}(\text{tx}, sk_{A, F})$  and  $\text{SIGN}(\text{tx}, sk_{\text{loc}, \text{virt}})$  to sigs // one of the two
     signatures may be empty, as some transactions don't need a signature by both
     keys. This is not a problem.
15:  ignore all messages except for (CHECK CHAIN FOR CLOSING) by  $R$ 
16:   $State \leftarrow \text{CLOSING}$ 
17:  send (SUBMIT, (tx, sigs)) to  $\mathcal{G}_{\text{Ledger}}$ 

18: On (CHECK CHAIN FOR CLOSING) by  $R \in \{\text{guest}, \text{funder}, \text{funder}\}$ :
19:   ensure  $State = \text{CLOSING}$ 
20:   send (READ) to  $\mathcal{G}_{\text{Ledger}}$  as  $R$  and assign reply to  $\Sigma$ 
21:   if  $R = \text{guest}$  then
22:      $pk_1 \leftarrow pk_{\text{left}, \text{guest}}; pk_2 \leftarrow pk_{\text{right}, \text{guest}}$ 
23:   else //  $R \in \{\text{funder}, \text{funder}\}$ 
24:      $pk_1 \leftarrow pk_{\text{loc}, \text{virt}}; pk_2 \leftarrow pk_{\text{rem}, \text{virt}}$ 
25:   end if
26:   if  $\Sigma$  has an unspent output that can be spent exclusively by a 2-of- $\{pk_1,$ 
      $pk_2\}$  multisig then // if there is a timelock, it must have expired
27:      $State \leftarrow \text{CLOSED}$ 
28:     output (CLOSED) to  $R$ 
29:   end if

```

Fig. 33.

Simulator \mathcal{S} – Pt. 1

- 1: On (OPEN, c_F , $pk_{A,out}$, $pk_{B,out}$, F , sig_F *Alice*) by $\mathcal{F}_{\text{Chan}}$: // both honest
- 2: simulate *Alice* receiving input (OPEN, c_F , $pk_{A,out}$, $pk_{B,out}$) by \mathcal{E}
- 3: ensure simulated *Alice* inputs (SUBMIT, (F' , $\text{sig}_{F'}$)) to $\mathcal{G}_{\text{Ledger}}$
- 4: input (SUBMIT, (F , sig_F)) to $\mathcal{G}_{\text{Ledger}}$

- 5: On (OPEN, c_F , $pk_{A,out}$, $pk_{B,out}$, $pk_{B,F}$, *Bob*) by $\mathcal{F}_{\text{Chan}}$: // *Alice* corrupted
- 6: send LN message (OPEN, $pk_{B,F}$) to *Alice* and relay reply to $\mathcal{F}_{\text{Chan}}$ **TODO:**
 change msg to fit LN, ensure *Alice* doesn't see a difference from real world

- 7: On (PAY, x , *Dave*) by $\mathcal{F}_{\text{Chan}}$:
- 8: **if** both channel parties are honest **then**
- 9: simulate *Dave* receiving input (PAY, x) by \mathcal{E}
- 10: ensure simulated *Dave* outputs (OK)
- 11: send (OK) to $\mathcal{F}_{\text{Chan}}$
- 12: **else if** only *Dave's* counterparty is corrupted **then** // else just relay to \mathcal{A}
- 13: simulate *Dave* receiving input (PAY, x) by \mathcal{E}
- 14: ensure simulated *Dave* outputs (OK)
- 15: extract the latest commitment transaction C and its signature by
 Dave's counterparty $\text{sig}_{\bar{D},C}$ from simulated *Dave's* state
- 16: send (C , $\text{sig}_{\bar{D},C}$) to $\mathcal{F}_{\text{Chan}}$
- 17: **end if**

- 18: On (FUND YOU, c , *Bob*, *Charlie*, *Alice*) by $\mathcal{F}_{\text{Chan}}$:
- 19: simulate *Alice* receiving input (FUND YOU, c , *Bob*) by *Charlie*
- 20: ensure simulated *Alice* outputs (OK) to *Charlie*
- 21: send (OK) to $\mathcal{F}_{\text{Chan}}$

- 22: On (FUND c , hops, sub_parties = (fundee, counterparty), outer_parties =
 (*Charlie*, *Dave*), funder = *Alice*, id) by $\mathcal{F}_{\text{Chan}}$:
- 23: add the message data to virtual_opening
- 24: simulate execution of line ?? of Fig. ?? with *Alice*// \mathcal{S} knows *Bob* (*Alice's*
 counterparty) through opening procedure
- 25: send (OK) to $\mathcal{F}_{\text{Chan}}$

- 26: On (ALLOW FUND, c , sub_parties, local_funder = L_i , id, $i \stackrel{?}{=} |\text{hops}|$) by
 $\mathcal{F}_{\text{Chan}}$'s *Alice* to *Charlie*:
- 27: simulate receiving message with *Charlie* by *Alice* and all subsequent
 communication
- 28: ensure the simulated *Charlie* sends (OK) to the simulated *Alice*
- 29: intercept this message and send it to $\mathcal{F}_{\text{Chan}}$'s *Alice*

Fig. 34.

Simulator \mathcal{S} – Pt. 2

- 1: On (IS OPEN SUCCESSFUL, id) by $\mathcal{F}_{\text{Chan}}$:
- 2: retrieve and remove from **virtual_opening** the data marked with id
- 3: simulate line ?? of Fig. ?? with *Alice* using this data
- 4: ensure *Alice* completes execution of VChan() successfully
- 5: send (OK) to $\mathcal{F}_{\text{Chan}}$

- 6: On (UPDATE TO VIRTUAL) by $\mathcal{F}_{\text{Chan}}$:
- 7: retrieve and remove from **virtual_opening** the data marked with id
- 8: simulate line ?? of Fig. ?? with *Alice* using this data
- 9: ensure *Alice* completes execution of VChan() successfully
- 10: extract from *Alice*’s state the new virtual funding TX V for pre-existing channel
- 11: extract from *Alice*’s state the new commitment TX C that spends the on-chain funding TX
- 12: send (V, C) to $\mathcal{F}_{\text{Chan}}$

- 13: On (FUND DONE, id) by $\mathcal{F}_{\text{Chan}}$ ’s *Alice* to *Charlie*:
- 14: simulate receiving message with *Charlie* by *Alice* and all subsequent communication
- 15: ensure the simulated *Charlie* sends (OK) to the simulated *Alice*
- 16: intercept this message and send it to $\mathcal{F}_{\text{Chan}}$ ’s *Alice*

Fig. 35.

1 Security Proof

When \mathcal{E} sends (FUND, c , hops, (fundee, counterparty), (*Charlie*, *Dave*), $pk_{VA,out}$, $pk_{VB,out}$) to *Alice* in the real world, lines ??-?? of Fig. ?? are executed and then control is handed over to the “fundee” ITI, which executes lines ??-?? of Fig. ?. This ITI will output (OK) if and only if line ?? of Fig. ? succeeds.

When \mathcal{E} sends (FUND, c , hops, (fundee, counterparty), (*Charlie*, *Dave*)) to *Alice* in the ideal world, lines ??-?? of Fig. ? are executed and then control is handed over to the functionality that controls the “fundee”, which executes lines ??-?? of Fig. ? and then hands control over to \mathcal{S} . The latter in turn simulates lines ??-?? of Fig. ?, thus following the exact same steps as in the real world, therefore it will send (OK) to $\mathcal{F}_{\text{Chan}}$ if and only if the simulated line ?? of Fig. ? succeeds. From this and the previous paragraph, we see that, up to this point, the two worlds are perfectly indistinguishable.

Moving on, in the ideal world subsequently lines ??-?? of Fig. ? are executed, which results in \mathcal{S} executing lines 22-25 of Fig. 34. During the latter steps, \mathcal{S} simulates executing line ?? of Fig. ? with *Alice*.

Similarly in the real world, *Alice* executes lines ?? and ?? of Fig. ?, therefore the two worlds still are perfectly indistinguishable.

The “for” loop of lines ??-?? of Fig. ?? is then executed in both the real and the ideal worlds. The message of line ?? results in the execution of lines ??-?? of Fig. ?? by L_i in both worlds: in the real world directly, in the ideal world simulated by \mathcal{S} .

In the ideal world, line ?? in Fig. ?? prompts \mathcal{S} to simulate line ?? of Fig. ?? with *Alice*, which is exactly the code that would be directly run by *Alice* in the real world. Therefore the two worlds remain perfectly indistinguishable.

The “for” loop of lines ??-?? of Fig. ?? is also perfectly indistinguishable in the two worlds. With argumentation similar to that of the previous “for” loop, we conclude that the FUND message does not induce any chance of distinguishability between the two worlds.

Theorem 1. *Assume that at the end of the execution, $\mathcal{G}_{\text{Ledger}}$ contains exactly one “groups” transaction that precedes all “funding” transactions and contains as payload a partition \mathcal{G} into groups of all VChan parties, with each group containing the parties that belong to the same (human) owner. Then the following holds:*

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

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

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

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

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

References