Protocol Π_{Chan}

1: On (BALANCE) by $\mathcal{E}$, act like $\mathcal{F}_{\text{Chan}}$ (Fig. 3, lines 2-3)

Fig. 1.

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 $\mathcal{F}_{\text{Chan}}$ – init, top up & corruption

1: On (CORRUPT) by P, addressed to *Alice*:
2: ensure $P \in \{\texttt{host\_alice}, \mathcal{A}\}$
3: `virtual_secrets` $\leftarrow \emptyset$
4: **for all** $(\_, \_, (\text{fundee}, \_), (\textit{Alice}, \_), \textit{vid}) \in$ `virtuals` **do**
5: send (CORRUPT) to fundee and ensure reply is (CORRUPTED, `secrets`)
6: append (`secrets`, *vid*) to `virtual_secrets`
7: **end for**
8: from now on, allow $\mathcal{A}$ to handle all *Alice*'s messages, i.e. act as a relay
9: **if** *Bob* is not corrupted **then**
10: from now on, handle all messages by *Bob* as Π_{Chan} (Fig. 1-**??**)
11: **end if**
12: **if** $P =$ `host_alice` **then**
13: output (CORRUPTED, (LN.SECRETS(*Alice*), `virtual_secrets`)) to `host_alice`
14: **else** // $P = \mathcal{A}$
15: send (CORRUPTED, (LN.SECRETS(*Alice*), `virtual_secrets`)) to $\mathcal{A}$
16: **end if**

Fig. 2.

Functionality $\mathcal{F}_{\text{Chan}}$ – base

1: On (BALANCE) by $\textit{Dave} \in \{\textit{Alice}, \textit{Bob}\}$:
2: ensure $\textit{State} \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$
3: output (BALANCE, $c_A, c_B, \text{locked}(A), \text{locked}(B)$) to *Dave*

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, $pk_{P,\text{out}}$):
4: ensure $State^P = \bot$
5: $State^P \leftarrow$ INIT
6: store $pk_{P,\text{out}}$
7: $(c_A, c_B, \texttt{locked}_A, \texttt{locked}_B) \leftarrow (0, 0, 0, 0)$
8: $(\texttt{paid\_out}, \texttt{paid\_in}) \leftarrow (\emptyset, \emptyset)$
9: output (INIT OK)

10: On (TOP UP):
11: ensure $P = Alice$ // activated party is the funder
12: ensure $State^P =$ INIT
13: $(sk_{P,\text{chain}}, pk_{P,\text{chain}}) \leftarrow$ KEYGEN()
14: input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign ouput to Σ
15: output (TOP UP TO, $pk_{P,\text{chain}}$)
16: **while** $\nexists \text{tx} \in \Sigma, c_{P,\text{chain}} : (c_{P,\text{chain}}, pk_{P,\text{chain}}) \in \text{tx.outputs}$ **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 ouput to Σ
20: **end while**
21: $State^P \leftarrow$ TOPPED UP
22: output (TOP UP OK, $c_{P,\text{chain}}$)

Fig. 4.

Process LN – methods used by VIRT

1: getCommTX(c_A, c_B, $pk_{A,F}$, $pk_{A,\text{out}}$, $pk_{A,R}$, $pk_{B,F}$, $pk_{B,\text{out}}$, $pk_{B,R}$, P):
2: $C_{P,i+1} \leftarrow$ TX {input: $(c_A + c_B, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $((c_A, (pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}), (c_B, pk_{B,\text{out}}))$}
3: **return** $C_{P,i+1}$

4: getCommKeys():
5: **return** $(pk_{A,F}, pk_{A,\text{out}}, pk_{A,R}), (pk_{B,F}, pk_{B,\text{out}}, pk_{B,R})$

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

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

Fig. 5.

Process LN.EXCHANGEOPENKEYS()

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

Fig. 6.

Process LN.PREPAREBASE()

1: **if** `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
2: $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}
3: $\texttt{host}_A \leftarrow \mathcal{G}_{\text{Ledger}}$
4: **if** ideal world **then**
5: $\texttt{host}_B \leftarrow \mathcal{G}_{\text{Ledger}}$
6: $State^B \leftarrow$ 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, `host_fundee`) to `fundee` by `hops`[-1].`right` // ignore any other message
11: input (FUND ACK) as `fundee` to `hops`[-1].`right` and expect output (FUNDED, `host_funder`) 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, $\texttt{host}_P$) // ignore any other message
14: **end if**
15: **end if**

Fig. 7.

Process LN.EXCHANGEOPENSIGS()

1: $C_{A,0} \leftarrow$ 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: $C_{B,0} \leftarrow$ 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: $\text{sig}_{A,C,0} \leftarrow$ SIGN$(C_{B,0}, sk_{A,F})$
4: $State^A \leftarrow$ WAITING FOR COMM SIG
5: send (FUNDING CREATED, $(c, pk_{A,\text{chain}})$, $\text{sig}_{A,C,0}$) to `fundee`
6: ensure $State^B =$ 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: $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}
9: **end if**
10: $C_{B,0} \leftarrow$ 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 VERIFY$(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F}) =$ True
12: $C_{A,0} \leftarrow$ 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: $\text{sig}_{B,C,0} \leftarrow$ SIGN$(C_{A,0}, sk_{B,F})$
14: **if** `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
15: $State^B \leftarrow$ WAITING TO CHECK FUNDING
16: **else** // opening virtual channel
17: $c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
18: $State^B \leftarrow$ OPEN
19: **end if**
20: reply (FUNDING SIGNED, $\text{sig}_{B,C,0}$)
21: ensure $State^A =$ WAITING FOR COMM SIG
22: ensure VERIFY$(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F}) =$ True

Fig. 8.

Process LN.COMMITBASE()

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

Fig. 9.

Process LN – external open messages for Bob

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

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

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 $State^A =$ TOPPED UP
5: ensure $c = c_{A,\text{chain}}$
6: **else** // opening virtual channel
7: ensure len(`hops`) ≥ 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: $State^B \leftarrow$ OPEN
20: **end if**
21: $c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
22: $State^A \leftarrow$ 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: $\text{sig}_{P,C,i+1} \leftarrow$ SIGN($C_{\bar{P},i+1}$) // kept by $\bar{P}$
3: send (UPDATE FORWARD, $\text{sig}_{P,C,i+1}$) to $\bar{P}$
4: // P refers to payer and $\bar{P}$ to payee both in local and remote code
5: $C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with $pk'_{P,F}$ and $pk'_{\bar{P},F}$ instead of $pk_{P,F}$ and $pk_{\bar{P},F}$ respectively
6: ensure VERIFY($C_{\bar{P},i+1}$, $\text{sig}_{P,C,i+1}$, $pk'_{P,F}$) = True
7: $C_{P,i+1} \leftarrow C_{P,i}$ with $pk'_{\bar{P},F}$ and $pk'_{P,F}$ instead of $pk_{\bar{P},F}$ and $pk_{P,F}$ respectively
8: $\text{sig}_{\bar{P},C,i+1} \leftarrow$ SIGN($C_{P,i+1}$, $sk'_{\bar{P},F}$) // kept by P
9: reply (UPDATE BACK, $\text{sig}_{\bar{P},C,i+1}$)
10: $C_{P,i+1} \leftarrow C_{P,i}$ with $pk'_{\bar{P},F}$ and $pk'_{P,F}$ instead of $pk_{\bar{P},F}$ and $pk_{P,F}$ respectively
11: ensure VERIFY($C_{P,i+1}$, $\text{sig}_{\bar{P},C,i+1}$, $pk'_{\bar{P},F}$) = True

Fig. 12.

Process LN – virtualise start and end

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

12: On output (HOSTS READY) by $\texttt{host}'_P$: // real or ideal world
13: ensure $State^P$ = WAITING FOR HOSTS READY
14: $State^P \leftarrow$ OPEN
15: $\texttt{host}_P \leftarrow \texttt{host}'_P$ // 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** len(**hops**) = 1 **then** // we are the last hop
19: output (FUNDED, $\texttt{host}_P$) 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, $\texttt{host}_P$) to **funder** // do not expect reply by **funder**
22: **end if**
23: reply (HOST ACK)

Fig. 13.

Process LN – virtualise hops

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

On input (VIRTUALISING, host_sibling, $(sk'_{P,F}, pk'_{P,F})$, $pk_{\text{sib},\bar{P},F}$, hops, fundee, c_{guest}, $c_{\text{sib,rem}}$, sib) by sibling:
  ensure $State^P$ = OPEN
  ensure $c_P - \texttt{locked}_P \geq c$
  ensure $c_{\text{sib,rem}} \geq c_P \wedge c_{\bar{P}} \geq c_{\text{sib}}$ // avoid value loss by griefing attack: one counterparty closes with old version, the other does not close
  $State^P \leftarrow$ VIRTUALISING
  $\texttt{locked}_P \leftarrow \texttt{locked}_P + c$
  define new VIRT ITI $\texttt{host}'_P$
  send (VIRTUALISING, $\texttt{host}'_P$, $pk'_{P,F}$, hops, fundee, c_{guest}) to hops[0].right and expect reply (VIRTUALISING ACK, $\texttt{host}'_{\bar{P}}$, $pk'_{\bar{P},F}$)
  ensure $pk'_{\bar{P},F}$ is different from $pk_{\bar{P},F}$ and all older $\bar{P}$'s funding public keys
  LN.UPDATEFORVIRTUAL()
  input (INIT, $\texttt{host}_P$, $\texttt{host}'_{\bar{P}}$, host_sibling, $(sk'_{P,F}, pk'_{P,F})$, $pk'_{\bar{P},F}$, $pk_{\text{sib},\bar{P},F}$, $(sk_{P,F}, pk_{P,F})$, $pk_{\bar{P},F}$, c_{guest}) to $\texttt{host}'_P$ and expect reply (HOST INIT OK)
  $State^P \leftarrow$ WAITING FOR REVOCATION
  output (VIRTUALISING ACK, $\texttt{host}'_P$, $pk'_{\bar{P},F}$) 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** $\texttt{host}_{\bar{P}} \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$ has a `host_sibling` **then** // we are intermediary channel
6: ensure $c_{\text{sib,rem}} \geq c_P - x \wedge c_{\bar{P}} + x \geq c_{\text{sib}}$ // 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: $R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.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: $R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: ($c_{\bar{P}}$, $pk_{P,\text{out}}$)}
3: ensure VERIFY($R_{P,i}$, $\text{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
4: $R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: (c_P, $pk_{\bar{P},\text{out}}$)}
5: $\text{sig}_{P,R,i} \leftarrow$ SIGN($R_{\bar{P},i}$, $sk_{P,R}$)
6: add x to `paid_out`
7: $c_P \leftarrow c_P - x$; $c_{\bar{P}} \leftarrow c_{\bar{P}} + x$; $i \leftarrow i + 1$
8: **if** $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge$ we have a `host_sibling` **then** // we are intermediary channel
9: input (EW BALANCE, c_P, $c_{\bar{P}}$) to $\texttt{host}_P$
10: relay message as input to `sibling` // run by VIRT
11: relay message as output to `guest` // run by VIRT
12: store new sibling balance and reply (NEW BALANCE OK)
13: output (NEW BALANCE OK) to `sibling` // run by VIRT
14: output (NEW BALANCE OK) to `guest` // run by VIRT
15: **end if**
16: send (REVOKE AND ACK, $\text{sig}_{P,R,i}$) to $\bar{P}$
17: $R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.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** $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$ has a `host_sibling` **then** // we are intermediary channel
22: input (NEW BALANCE, $c_{\bar{P}}$, c_P) to $\texttt{host}_{\bar{P}}$
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):

ensure $State^{P} = \text{OPEN} \wedge c_P \geq x$
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge P$ has a `host_sibling` **then** // we are intermediary channel
    ensure $c_{\text{sib,rem}} \geq c_P - x \wedge c_{\bar{P}} + x \geq c_{\text{sib}}$ // avoid value loss by griefing attack: one counterparty closes with old version, the other does not close
end if
if ideal world **then**
    send (PAY, x) to $\mathcal{A}$ and expect reply (PAY OK)
    ensure $State^{\bar{P}} = \text{OPEN}$
    $c_P \leftarrow c_B - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x; i \leftarrow i + 1$
    if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge P$ has a `host_sibling` **then** // we are intermediary channel
        input (NEW BALANCE, c_P, $c_{\bar{P}}$) to $\texttt{host}_P$
        relay message as input to `sibling` // run by VIRT
        relay message as output to `guest` // run by VIRT
        store new sibling balance and reply (NEW BALANCE OK)
        output (NEW BALANCE OK) to `sibling` // run by VIRT
        output (NEW BALANCE OK) to `guest` // run by VIRT
    end if
else
    LN.SIGNATURESROUNDTRIP()
    LN.REVOCATIONSTRIP()
end if
// No output is given to the caller, this is intentional

Fig. 17.

Process LN – On (CLOSE):

1: TODO: select more states (e.g. VIRTUALISING) from which one can close – thus avoid DoS
2: ensure $State^P = \text{OPEN}$
3: **if** $\texttt{hops} \neq \mathcal{G}_{\text{Ledger}}$ **then** // we have a virtual channel
4: $State^P \leftarrow$ HOST CLOSING
5: input (CLOSE) to $\texttt{hops}[0].\texttt{left}$ and keep relaying inputs (CHECK CHAIN FOR CLOSING) to $\texttt{hops}[0].\texttt{left}$ until receiving output (CLOSED)
6: **end if**
7: input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign output to Σ
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: ensure LN.SUBMITREVOCATION(j) returns (OK)
12: **else** // counterparty is idle
13: **while** $\nexists$ unspent $\texttt{output} \in \Sigma$ 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 Σ
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: $State^P \leftarrow$ CLOSING
20: input (SUBMIT, $(C_{P,i}, \text{sig}_{P,C,i}, \text{sig}'_{P,C,i})$) to $\mathcal{G}_{\text{Ledger}}$
21: **while** $C_{P,i} \notin \Sigma$ **do**
22: wait for input (CHECK CLOSED) // ignore other messages
23: input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign output to Σ
24: **end while**
25: // provably reachable – c.f. TODO: ref
26: **end if**
27: $State^P \leftarrow$ CLOSED
28: output (CLOSED)

Fig. 18.

Process VIRT

On input (INIT, `host`$_P$, $\bar{P}$, `sibling`, ($sk_{\text{loc,virt}}$, $pk_{\text{loc,virt}}$), $pk_{\text{rem,virt}}$, $pk_{\text{sib,rem,virt}}$, ($sk_{\text{loc},F}$, $pk_{\text{loc},F}$), $pk_{\text{rem},F}$, c_{guest}) by `guest`:
  store message contents and `guest` // `sibling`, $pk_{\text{sib},\bar{P},F}$ are missing for edge nodes
  output (HOST INIT OK) to `guest`

On input (HOST ME, `funder`, `fundee`, $\bar{P}$, `host`$_P$, c_{guest}, $pk_{A,V}$, $pk_{B,V}$, ($sk_{\text{loc,virt}}$, $pk_{\text{loc,virt}}$), ($sk_{\text{loc},F}$, $pk_{\text{loc},F}$), $pk_{\text{rem},F}$, $pk_{\text{rem,virt}}$) by `guest`:
  ensure VIRT.CIRCULATEKEYSANDCOINS() returns (OK)
  ensure VIRT.CIRCULATEVIRTUALSIGS() returns (OK)
  ensure VIRT.CIRCULATEFUNDINGSIGS() returns (OK)
  ensure VIRT.CIRCULATEREVOCATIONS() returns (OK)
  output (HOSTS READY) to `guest`

CIRCULATEKEYSANDCOINS(`left_data`):
  if `left_data` is given as argument **then** // we are not `host_funder`
    if we have a `sibling` **then** // we are not `host_fundee`
      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`
      store input as `left_data`
      parse `left_data` as `far_left_data`, ($sk_{\text{loc,virt}}$, $pk_{\text{loc,virt}}$), ($sk_{\text{sib},F}$, $pk_{\text{sib},F}$), $pk_{\text{sib,rem},F}$, c_{sib}, $c_{\text{sib,rem}}$ // remove parentheses as necessary
      call VIRT.CIRCULATEKEYSANDCOINS(`left_data`) of $\bar{P}$ and assign returned value to `right_data`
      parse `right_data` as `far_right_data`, $pk_{\text{rem,virt}}$
      output (KEYS AND COINS BACK, `right_data`, ($sk_{\text{loc},F}$, $pk_{\text{loc},F}$), $pk_{\text{rem},F}$, c_P, $c_{\bar{P}}$)
      store output as `right_data`
      parse `right_data` as `far_right_data`, ($sk_{\text{sib},F}$, $pk_{\text{sib},F}$), $pk_{\text{sib,rem},F}$, c_{sib}, $c_{\text{sib,rem}}$
      return (`right_data`, $pk_{\text{loc,virt}}$)
    else // we are `host_fundee`
      return $pk_{\text{loc,virt}}$
    end if
  else // we are `host_funder`
    call VIRT.CIRCULATEKEYSANDCOINS($pk_{\text{loc,virt}}$) of $\bar{P}$ and assign returned value to `right_data`
    return (OK)
  end if

Fig. 19.

Process VIRT

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

Fig. 20.

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

Fig. 21.

Process VIRT.SIBLINGSIGS()

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

Fig. 22.

Process VIRT.INTERMEDIARYSIGS()

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

Fig. 23.

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

if $\text{sigs}_{\text{byLeft}}$ is given as argument **then** // we are not `host_funder`
  if we have a `sibling` **then** // we are not `host_fundee`
    return VIRT.INTERMEDIARYSIGS()
  else // we are `host_fundee`
    $\text{TX}_{\text{loc,none}} \leftarrow$ VIRT.GETEDGETXS(c_{guest}, c_P, $c_{\bar{P}}$, $pk_{\text{loc},F}$, $pk_{\text{rem},F}$, $pk_{\text{loc,virt}}$, $pk_{\text{rem,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, False)
    for each hop apart from the first and ours ($i \in [2, \ldots, n-1]$) **do** // hop data encoded in `left_data`
      extract data needed for GETMIDTXS() from `left_data` and assign it to $\texttt{data}_i$ and $\{pk_{\text{sec},i}\}_{i\in 1\ldots n}$ // $\{pk_{\text{sec},i}\}_{i\in 1\ldots n}$ contains each party's $pk_{i,\text{virt}}$
      $(\text{TX}_{i,\text{none}}, \text{TX}_{i,\text{left}}, \text{TX}_{i,\text{right}}, \text{TX}_{i,\text{both}}) \leftarrow$ VIRT.GETMIDTXS($\texttt{data}_i$, $\{pk_{\text{sec},i}\}_{i\in 1\ldots n}$)
      add SIGN($\text{TX}_{i,\text{right}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) and SIGN($\text{TX}_{i,\text{both}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) to $\text{sigs}_{\text{toLeft}}$ // if i-th hop is adjacent, 2 signatures will be produced by each SIGN() invocation: one for the `guest_all` and one for the $2 \wedge$ `right_virt_checked` input
      if $i = n-1$ **then** // hop is our previous
        add SIGN($\text{TX}_{i,\text{left}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) to $\text{sigs}_{\text{toLeft}}$
      end if
    end for
    return $\text{sigs}_{\text{toLeft}}$
  end if
else // we are `host_funder`
  for each hop apart from the last and ours ($i \in [2, \ldots, n-1]$) **do** // hop data encoded in `right_data`
    extract data needed for GETMIDTXS() from `right_data` and assign it to $\texttt{data}_i$ and $\{pk_{\text{sec},i}\}_{i\in 1\ldots n}$ // $\{pk_{\text{sec},i}\}_{i\in 1\ldots n}$ contains each party's $pk_{i,\text{virt}}$
    $(\text{TX}_{i,\text{none}}, \text{TX}_{i,\text{left}}, \text{TX}_{i,\text{right}}, \text{TX}_{i,\text{both}}) \leftarrow$ VIRT.GETMIDTXS($\texttt{data}_i$, $\{pk_{\text{sec},i}\}_{i\in 1\ldots n}$)
    add SIGN($\text{TX}_{i,\text{right}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) and SIGN($\text{TX}_{i,\text{both}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) to $\text{sigs}_{\text{toRight}}$ // if i-th hop is adjacent, 2 signatures will be produced by each SIGN() invocation: one for the `guest_all` and one for the $2 \wedge$ `right_virt_checked` input
    if $i = 2$ **then** // hop is our next
      add SIGN($\text{TX}_{i,\text{left}}$, $sk_{\text{loc,virt}}$, ANYPREVOUT) to $\text{sigs}_{\text{toRight}}$
    end if
  end for
  call VIRT.CIRCULATEVIRTUALSIGS($\text{sigs}_{\text{toRight}}$) of P and assign output to $\text{sigs}_{\text{byRight}}$
  $\text{TX}_{\text{loc,none}} \leftarrow$ VIRT.GETEDGETXS(c_{guest}, c_P, $c_{\bar{P}}$, $pk_{\text{loc},F}$, $pk_{\text{rem},F}$, $pk_{\text{loc,virt}}$, $pk_{\text{rem,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, True)
  return (OK)
end if

Fig. 24.

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

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

Fig. 25.

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. 26.

Process VIRT – close

On input (CLOSE) by $P \in$ `guests`:
  if *State* = CLOSED **then**
    output (CLOSED) to P
  end if
  ensure *State* = OPEN
  if `host` $\neq \mathcal{G}_{\text{Ledger}}$ **then** // `host` is a VIRT
    ignore all messages except for output (CLOSED) by `host`. Also relay to `host` any (CHECK CHAIN FOR CLOSING) input received
    input (CLOSE) to `host`
  end if
  // if we have a `host`, continue from here on output (CLOSED) by it
  send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to `chain`
  let `tx` be the unique valid TX for `chain` among ($\text{TX}_{\text{loc,none}}$, $\text{TX}_{\text{loc,left}}$, $\text{TX}_{\text{loc,right}}$, $\text{TX}_{\text{loc,both}}$) // if we are not an intermediary, only the first exists
  let `sigs` be the corresponding set of signatures among ($\text{sigs}_{\text{loc,none}}$, $\text{sigs}_{\text{loc,left}}$, $\text{sigs}_{\text{loc,right}}$, $\text{sigs}_{\text{loc,both}}$)
  add SIGN(`tx`, $sk_{A,F}$) and SIGN(`tx`, $sk_{\text{loc,virt}}$) to `sigs` // one of the two signatures may be empty, as some transactions don't need a signature by both keys. This is not a problem.
  ignore all messages except for (CHECK CHAIN FOR CLOSING) by $\mathcal{E}$
  State $\leftarrow$ CLOSING
  send (SUBMIT, (`tx`, `sigs`)) to $\mathcal{G}_{\text{Ledger}}$

On (CHECK CHAIN FOR CLOSING) by $P \in$ `guests`:
  ensure *State* = CLOSING
  send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to `chain`
  if `tx` $\in$ `chain` **then**
    State $\leftarrow$ CLOSED
    output (CLOSED) to P
  end if

Fig. 27.

Functionality $\mathcal{G}_{\text{Trust}}$

On (SET TRUSTS, G) by $\mathcal{E}$:
    ensure $State = \bot$
    ensure G is a directed forest where the nodes are ITM identifiers
    store G
    $State \leftarrow \top$

On (IS PARENT, id_1, id_2) by P
    if id_1 is the parent of id_2 in G **then**
        send (IS PARENT, id_1, id_2, true) to P
    else
        send (IS PARENT, id_1, id_2, false) to P
    end if

Simulator $\mathcal{S}$– Pt. 1

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

On (OPEN, c_F, $pk_{A,\text{out}}$, $pk_{B,\text{out}}$, $pk_{B,F}$, *Bob*) by $\mathcal{F}_{\text{Chan}}$: // *Alice* corrupted
    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

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

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

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

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

Fig. 28.

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. 29.

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. 28. 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:*

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

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

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

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

$$\text{top-up}(P) = \begin{cases} c_{\text{on}}, & \text{as determined on message (CHECK TOP UP),} \\ & \text{if such a message was handled} \\ 0, & \text{otherwise} \end{cases}$$

$$\text{pay-in}(m, P) = \begin{cases} x, & \text{if message } m \text{ updated the channel to} \\ & \text{a state in which } P \text{ had } x \text{ more coins} \textit{TODO: improve prev} \\ 0, & \text{otherwise} \end{cases}$$

$$\text{pay-out}(m, P) = \begin{cases} x, & \text{if } m = (\text{PAY}, x) \text{ was received by } P \text{ and} \\ & P \text{ output (PAY SUCCESS) as a result} \\ 0, & \text{otherwise} \end{cases}$$

References