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

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

- On receiving input (`msg`) by $\mathcal{E}$ to $P \in \{Alice, Bob\}$, handle it according to the corresponding rule in Fig. 2, 3, or 4 (if any) and subsequently send (RELAY, `msg`, P, $\mathcal{E}$, input) $\mathcal{A}$. // all messages by $\mathcal{E}$ are relayed to $\mathcal{A}$
- On receiving (`msg`) by $R \neq \mathcal{E}$ to $P \in \{Alice, Bob\}$ by means of `mode` $\in \{\text{input}, \text{output}, \text{network}\}$, send (RELAY, `msg`, P, R, `mode`) to $\mathcal{A}$. // all messages by machines other than $\mathcal{E}$ are relayed to $\mathcal{A}$
- On receiving (RELAY, `msg`, P, R, `mode`) by $\mathcal{A}$ (`mode` $\in \{\text{input}, \text{output}, \text{network}\}$, $P \in \{Alice, Bob\}$), relay `msg` to R as P by means of `mode`. // $\mathcal{A}$ fully controls outgoing messages by $\mathcal{F}_{\text{Chan}}$
- On receiving (INFO, `msg`) by $\mathcal{A}$, handle (`msg`) according to the corresponding rule in Fig. 2, 3, or 4 (if any). After handling the message or after an "ensure" fails, send (HANDLED, `msg`) to $\mathcal{A}$. // (INFO, `msg`) messages by $\mathcal{S}$ always return control to $\mathcal{S}$ without any side-effect to any other ITI, except if $\mathcal{F}_{\text{Chan}}$ halts

Fig. 1.

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

On initalisation:
  $pk_P \leftarrow \bot$; $\texttt{host}_P \leftarrow \bot$; $\texttt{enabler}_P \leftarrow \bot$; $\texttt{balance}_P \leftarrow 0$;
  $State_P \leftarrow$ UNINIT

On (BECAME CORRUPTED OR NEGLIGENT, P) by $\mathcal{A}$ or on output (ENABLER USED REVOCATION) by $\texttt{host}_P$ when in any state:
  $State_P \leftarrow$ IGNORED

On (INIT, pk) to P by $\mathcal{E}$ when $State_P =$ UNINIT:
  $pk_P \leftarrow pk$
  $State_P \leftarrow$ INIT

On (OPEN, x, $\mathcal{G}_{\text{Ledger}}$, ...) to *Alice* by $\mathcal{E}$ when $State_A =$ INIT:
  store x
  $State_A \leftarrow$ TENTATIVE BASE OPEN

On (BASE OPEN) by $\mathcal{A}$ when $State_A =$ TENTATIVE BASE OPEN:
  $\texttt{balance}_A \leftarrow x$
  $State_A \leftarrow$ OPEN

On (BASE OPEN) by $\mathcal{A}$ when $State_B =$ INIT:
  $State_B \leftarrow$ OPEN

On (OPEN, x, $\texttt{hops} \neq \mathcal{G}_{\text{Ledger}}$, ...) to *Alice* by $\mathcal{E}$ when $State_A =$ INIT:
  store x
  $\texttt{enabler}_A \leftarrow \texttt{hops}[0].\texttt{left}$
  $State_A \leftarrow$ PENDING VIRTUAL OPEN

On output (FUNDED, $\texttt{host}$, ...) to *Alice* by $\texttt{enabler}_A$ when $State_A =$ PENDING VIRTUAL OPEN:
  $\texttt{host}_A \leftarrow \texttt{host}[0].\texttt{left}$
  $State_A \leftarrow$ TENTATIVE VIRTUAL OPEN

On output (FUNDED, $\texttt{host}$, ...) to *Bob* by ITI $R \in \{\mathcal{F}_{\text{Chan}}, \text{LN}\}$ when $State_B =$ INIT:
  $\texttt{enabler}_B \leftarrow R$
  $\texttt{host}_B \leftarrow \texttt{host}$
  $State_B \leftarrow$ TENTATIVE VIRTUAL OPEN

On (VIRT OPEN) by $\mathcal{A}$ when $State_P =$ TENTATIVE VIRTUAL OPEN:
  if $P = Alice$ **then** $\texttt{balance}_P \leftarrow x$
  $State_P \leftarrow$ OPEN

Fig. 2.

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

1: On ($\textsc{pay}$, x) by $\mathcal{E}$ when $State_P = \textsc{open}$: // P pays $\bar{P}$
2: store x
3: $State_P \leftarrow \textsc{tentative pay}$

4: On ($\textsc{pay}$) by $\mathcal{A}$ when $State_P = \textsc{tentative pay}$: // P pays $\bar{P}$
5: $State_P \leftarrow (\textsc{sync pay}, x)$

6: On ($\textsc{get paid}$, x) by $\mathcal{E}$ when $State_P = \textsc{open}$: // $\bar{P}$ pays P
7: store x
8: $State_P \leftarrow \textsc{tentative get paid}$

9: On ($\textsc{pay}$) by $\mathcal{A}$ when $State_P = \textsc{tentative get paid}$: // $\bar{P}$ pays P
10: $State_P \leftarrow (\textsc{sync get paid}, x)$

11: When $State_P = (\textsc{sync pay}, x)$:
12: **if** $State_{\bar{P}} \in \{\textsc{ignored}, (\textsc{sync get paid}, x)\}$ **then**
13: $\texttt{balance}_P \leftarrow \texttt{balance}_P - x$
14: // if $\bar{P}$ honest, this state transition happens simultaneously with l. 21
15: $State_P \leftarrow \textsc{open}$
16: **end if**

17: When $State_P = (\textsc{sync get paid}, x)$:
18: **if** $State_{\bar{P}} \in \{\textsc{ignored}, (\textsc{sync pay}, x)\}$ **then**
19: $\texttt{balance}_P \leftarrow \texttt{balance}_P + x$
20: // if $\bar{P}$ honest, this state transition happens simultaneously with l. 15
21: $State_P \leftarrow \textsc{open}$
22: **end if**

23: On ($\textsc{close}$) by $\mathcal{E}$ when $State_P = \textsc{open}$:
24: $State_P \leftarrow \textsc{closing}$

25: On ($\textsc{close}$, P) by $\mathcal{A}$ when $State \in \{\textsc{open}, \textsc{closing}\}$:
26: input ($\textsc{read}$) to $\mathcal{G}_{\text{Ledger}}$ as P and assign ouput to Σ
27: $\texttt{coins}_P \leftarrow$ sum of coins exclusively spendable by pk_P in Σ
28: **if** $\texttt{coins}_P \geq \texttt{balance}_P$ **then**
29: $State_P \leftarrow \textsc{closed}$
30: **else** // balance security is broken
31: halt
32: **end if**

Fig. 3.

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

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

4: When $State_P =$ PENDING FUND:
5: **if** we intercept the command "define new VIRT ITI `host`" by $\mathcal{A}$ **then**
6: store `host`
7: $State_P \leftarrow$ TENTATIVE FUND
8: continue executing $\mathcal{A}$'s command
9: **end if**

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

12: When $State_P =$ OPEN:
13: **if** we intercept the command "define new VIRT ITI `host`" by $\mathcal{A}$ **then**
14: store `host`
15: $State_P \leftarrow$ TENTATIVE HELP FUND
16: continue executing $\mathcal{A}$'s command
17: **end if**

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

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

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

Fig. 4.

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

- On receiving (RELAY, `in_msg`, P, R, `in_mode`) by $\mathcal{F}_{\text{Chan}}$ (`in_mode` $\in$ {input, output, network}, $P \in \{Alice, Bob\}$), handle (`in_msg`) with the simulated party P as if it was received from R by means of `in_mode`. In case simulated P does not exist yet, initialise it as an LN ITI. If there is a resulting message `out_msg` that is to be sent by simulated P to R' by means of `out_mode` $\in$ {input, output, network}, send (RELAY, `out_msg`, P, R', `out_mode`) to $\mathcal{F}_{\text{Chan}}$.
- On receiving by $\mathcal{F}_{\text{Chan}}$ a message to be sent by P to R via the network, carry on with this action (i.e. send this message via the internal $\mathcal{A}$).
- Relay any other incoming message to the internal $\mathcal{A}$ unmodified.
- On receiving a message (`msg`) by the internal $\mathcal{A}$, if it is addressed to one of the parties that correspond to $\mathcal{F}_{\text{Chan}}$, handle the message internally with the corresponding simulated party. Otherwise relay the message to its intended recipient unmodified. // Other recipients are $\mathcal{E}$, $\mathcal{G}_{\text{Ledger}}$ or parties unrelated to $\mathcal{F}_{\text{Chan}}$

Given that $\mathcal{F}_{\text{Chan}}$ relays all messages and that we simulate the real-world machines that correspond to $\mathcal{F}_{\text{Chan}}$, the simulation is perfectly indistinguishable from the real world.

Fig. 5.

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

- "P" refers one of the parties that correspond to $\mathcal{F}_{\text{Chan}}$.
- When an action in this Figure interrupts an ITI simulation, continue simulating from the interruption location once action is over/$\mathcal{F}_{\text{Chan}}$ hands control back.

On (CORRUPT) by $\mathcal{A}$, addresed to P:
    // After executing this code, deliver (CORRUPT) to simulated P as detailed in Fig. 5. Given that $\mathcal{F}_{\text{Chan}}$ returns control directly to us after it handles this message, we will always deliver (CORRUPT) successfully.
    send (INFO, BECAME CORRUPTED OR NEGLIGENT, P) to $\mathcal{F}_{\text{Chan}}$

When simulated P sets its internal variable `negligent` to True (Fig. 8, l. 26/Fig. 7, l. 7):
    send (INFO, BECAME CORRUPTED OR NEGLIGENT, P) to $\mathcal{F}_{\text{Chan}}$

When simulated honest *Alice* receives (OPEN, x, $\dots$) by $\mathcal{E}$:
    store x // will be used to inform $\mathcal{F}_{\text{Chan}}$ once the channel is open

When the last of the honest simulated $\mathcal{F}_{\text{Chan}}$'s parties moves to the OPEN state for the first time (Fig. 11, l. 19/Fig. 13, l. 5/Fig. 14, l. 18):
    if x is undefined **then** $x \leftarrow 0$ // x is already defined if *Alice* is honest
    send (INFO, OPEN, x) to $\mathcal{F}_{\text{Chan}}$

When (both $\mathcal{F}_{\text{Chan}}$'s simulated parties are honest and P completes sending a payment of value c (Fig. 19, l. 6) and $\bar{P}$ completes receiving that payment (Fig. 19, l. 19)), or (when only S is honest and (completes either receiving ($\bar{P} \leftarrow S$, $P \leftarrow \bar{S}$) or sending ($P \leftarrow S$, $\bar{P} \leftarrow \bar{S}$) a payment of value c)): // also send this message if both parties are honest when Fig. 19, l. 6 is executed by one party, but its counterparty is corrupted before executing Fig. 19, l. 19
    send (INFO, PAY, c, P, $\bar{P}$) to $\mathcal{F}_{\text{Chan}}$

When honest simulated P executes Fig. 16, l. 20: // if $\bar{P}$ is honest, it has already moved to the new host, (Fig 36, l. 7 and 21), so lifting to next layer is complete
    extract c_{guest} from executed Fig. 16, l. 20
    send (INFO, FUND, c_{guest}, P) to $\mathcal{F}_{\text{Chan}}$

When one of the honest simulated $\mathcal{F}_{\text{Chan}}$'s parties P moves to the CLOSED state (Fig. 22, l. 8/Fig. 25, l. 27):
    send (INFO, CLOSE, P) to $\mathcal{F}_{\text{Chan}}$

Fig. 6.

Process LN – init

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

On (INIT, $pk_{P,\text{out}}$):
    ensure $State = \perp$
    $State \leftarrow$ INIT
    store $pk_{P,\text{out}}$
    $(c_A, c_B, \texttt{locked}_A, \texttt{locked}_B) \leftarrow (0, 0, 0, 0)$
    $(\texttt{paid\_out}, \texttt{paid\_in}) \leftarrow (\emptyset, \emptyset)$
    `negligent` $\leftarrow$ False
    `last_poll` $\leftarrow \perp$
    output (INIT OK)

On (TOP UP):
    ensure $P = Alice$ // activated party is the funder
    ensure $State =$ INIT
    $(sk_{P,\text{chain}}, pk_{P,\text{chain}}) \leftarrow$ KEYGEN()
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    output (TOP UP TO, $pk_{P,\text{chain}}$)
    while $\nexists \text{tx} \in \Sigma, c_{P,\text{chain}} : (c_{P,\text{chain}}, pk_{P,\text{chain}}) \in \text{tx.outputs}$ **do**
        // while waiting, all other messages by P are ignored
        wait for input (CHECK TOP UP)
        input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    end while
    $State \leftarrow$ TOPPED UP
    output (TOP UP OK, $c_{P,\text{chain}}$)

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

Fig. 7.

Process LN – methods used by VIRT

REVOKEPREVIOUS():

  ensure $\mathit{State} \in$ WAITING FOR (OUTBOUND) REVOCATION

  $R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.P, output: ($C_{P,i}$.outputs.P.value, $pk_{\bar{P},\mathrm{out}}$)}

  $\mathrm{sig}_{A,R,i} \leftarrow$ SIGN($R_{\bar{P},i}$, $sk_{P,R}$)

  if $\mathit{State} =$ WAITING FOR REVOCATION **then**

    $\mathit{State} \leftarrow$ WAITING FOR INBOUND REVOCATION

  else // $\mathit{State} =$ WAITING FOR OUTBOUND REVOCATION

    $i \leftarrow i + 1$

    $\mathit{State} \leftarrow$ WAITING FOR HOSTS READY

  end if

  $\mathtt{host}_P \leftarrow \mathtt{host}'_P$ // forget old host, use new host instead

  $\mathtt{layer} \leftarrow \mathtt{layer} + 1$

  return $\mathrm{sig}_{P,R,i}$

PROCESSREMOTEREVOCATION($\mathrm{sig}_{\bar{P},R,i}$):

  ensure $\mathit{State} =$ WAITING FOR (INBOUND) REVOCATION

  $R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: ($C_{\bar{P},i}$.outputs.$\bar{P}$.value, $pk_{P,\mathrm{out}}$)}

  ensure VERIFY($R_{P,i}$, $\mathrm{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True

  if $\mathit{State} =$ WAITING FOR REVOCATION **then**

    $\mathit{State} \leftarrow$ WAITING FOR OUTBOUND REVOCATION

  else // $\mathit{State} =$ WAITING FOR INBOUND REVOCATION

    $i \leftarrow i + 1$

    $\mathit{State} \leftarrow$ WAITING FOR HOSTS READY

  end if

  return (OK)

NEGLIGENT():

  $\mathtt{negligent} \leftarrow$ True

  return (OK)

Fig. 8.

Process LN.EXCHANGEOPENKEYS()

$(sk_{A,F}, pk_{A,F}) \leftarrow$ KEYGEN(); $(sk_{A,R}, pk_{A,R}) \leftarrow$ KEYGEN()
State $\leftarrow$ WAITING FOR OPENING KEYS
send (OPEN, c, `hops`, $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$) to `fundee`
// colored code is run by honest `fundee`. Validation is implicit
ensure we run the code of *Bob*
ensure *State* = INIT
store $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$
$(sk_{B,F}, pk_{B,F}) \leftarrow$ KEYGEN(); $(sk_{B,R}, pk_{B,R}) \leftarrow$ KEYGEN()
if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
    `layer` $\leftarrow 0$
    State $\leftarrow$ WAITING FOR COMM SIG
else // opening virtual channel
    State $\leftarrow$ WAITING FOR CHECK KEYS
end if
reply (ACCEPT CHANNEL, $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$)
ensure *State* = WAITING FOR OPENING KEYS
store $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$
State $\leftarrow$ OPENING KEYS OK

Fig. 9.

Process LN.PREPAREBASE()

if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
    $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}
    $\texttt{host}_P \leftarrow \mathcal{G}_{\text{Ledger}}$
    `layer` $\leftarrow 0$
else // opening virtual channel
    input (FUND ME, *Alice*, *Bob*, `hops`, c, $pk_{A,F}$, $pk_{B,F}$) to `hops`[0].`left` and expect output (FUNDED, $\texttt{host}_P$, `funder_layer`) // ignore any other message
    `layer` $\leftarrow$ `funder_layer`
end if

Fig. 10.

Process LN.EXCHANGEOPENSIGS()

// $s = (2 + \lceil \texttt{maxTime}_{\texttt{window}} + \frac{\texttt{Delay}}{2} / \texttt{minTime}_{\texttt{window}} \rceil)\texttt{windowSize}$, where $\texttt{maxTime}_{\texttt{window}}$, $\texttt{Delay}$, $\texttt{minTime}_{\texttt{window}}$ and $\texttt{windowSize}$ are defined in Proposition ?? TODO: recheck and include proposition
$C_{A,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, (pk_{A,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}
$C_{B,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, pk_{A,\text{out}})$, $(0, (pk_{B,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}
$\text{sig}_{A,C,0} \leftarrow \text{SIGN}(C_{B,0}, sk_{A,F})$
$State \leftarrow$ WAITING FOR COMM SIG
send (FUNDING CREATED, $(c, pk_{A,\text{chain}})$, $\text{sig}_{A,C,0}$) to **fundee**
ensure $State =$ WAITING FOR COMM SIG // if opening virtual channel, we have received (FUNDED, host_fundee) by hops[-1].right (Fig 13, l. 10)
if hops $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
    $F \leftarrow$ TX {input: $(c, pk_{A,\text{chain}})$, output: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$}
end if
$C_{B,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, pk_{A,\text{out}})$, $(0, (pk_{B,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}
ensure $\text{VERIFY}(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F}) = \text{True}$
$C_{A,0} \leftarrow$ TX {input: $(c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $(c, (pk_{A,\text{out}} + (t+s)) \vee 2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}
$\text{sig}_{B,C,0} \leftarrow \text{SIGN}(C_{A,0}, sk_{B,F})$
if hops $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
    $State \leftarrow$ WAITING TO CHECK FUNDING
else // opening virtual channel
    $c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
    $State \leftarrow$ OPEN
end if
reply (FUNDING SIGNED, $\text{sig}_{B,C,0}$)
ensure $State =$ WAITING FOR COMM SIG
ensure $\text{VERIFY}(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F}) = \text{True}$

Fig. 11.

Process LN.COMMITBASE()

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

Fig. 12.

Process LN – external open messages for *Bob*

On input (CHECK FUNDING):
    ensure $State =$ WAITING TO CHECK FUNDING
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
    if $F \in \Sigma$ **then**
        $State \leftarrow$ OPEN
        reply (OPEN OK)
    end if

On output (FUNDED, `host`$_P$, `funder_layer`) by `hops`[-1].`right`:
    ensure $State =$ WAITING FOR FUNDED
    store `host`$_P$ // we will talk directly to `host`$_P$
    `layer` $\leftarrow$ `funder_layer`
    $State \leftarrow$ WAITING FOR COMM SIG
    reply (FUND ACK)

On output (CHECK KEYS, (pk_1, pk_2)) by `hops`[-1].`right`:
    ensure $State =$ WAITING FOR CHECK KEYS
    ensure $pk_1 = pk_{A,F} \wedge pk_2 = pk_{B,F}$
    $State \leftarrow$ WAITING FOR FUDNED
    reply (KEYS OK)

Fig. 13.

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

// `fundee` is *Bob*
ensure we run the code of *Alice* // activated party is the funder
if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
    ensure *State* = TOPPED UP
    ensure $c = c_{A,\text{chain}}$
else // opening virtual channel
    ensure $\text{len}(\texttt{hops}) \geq 2$ // cannot open a virtual over 1 channel
end if
LN.EXCHANGEOPENKEYS()
LN.PREPAREBASE()
LN.EXCHANGEOPENSIGS()
if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then**
    LN.COMMITBASE()
end if
input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
`last_poll` $\leftarrow |\Sigma|$
$c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
State $\leftarrow$ OPEN
output (OPEN OK, c, `fundee`, `hops`)

Fig. 14.

Process LN.UPDATEFORVIRTUAL()

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

Fig. 15.

Process LN – virtualise start and end

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

On output (HOSTS READY) by $\texttt{host}_P$: // $\texttt{host}_P$ is the new host, renamed in Fig. 8, l. 12
  ensure $State =$ WAITING FOR HOSTS READY
  $State \leftarrow$ OPEN
  move $pk_{P,F}$, $pk_{\bar{P},F}$ to list of old funding keys
  $(sk_{P,F}, pk_{P,F}) \leftarrow (sk'_{P,F}, pk'_{P,F})$; $pk_{\bar{P},F} \leftarrow pk'_{\bar{P},F}$
  if len(`hops`) $= 1$ **then** // we are the last hop
    output (FUNDED, $\texttt{host}_P$, `layer`) to `fundee` and expect reply (FUND ACK)
  else if we have received input FUND ME just before we moved to the VIRTUALISING state **then** // we are the first hop
    $c_P \leftarrow c_P - c_{\text{guest}}$
    output (FUNDED, $\texttt{host}_P$, `layer`) to `funder` // do not expect reply by `funder`
  end if
  reply (HOST ACK)

On output (SIGN TXS, TXs) by $\texttt{host}'_P$:
  sigs $\leftarrow \emptyset$
  for TX in TXs **do**
    add SIGN(TX, $sk_{P,F}$, ANYPREVOUT) to sigs
  end for
  reply (TXS SIGNED, sigs)

Fig. 16.

Process LN – virtualise hops

On (VIRTUALISING, $\texttt{host}'_{\bar{P}}$, $pk'_{\bar{P},F}$, $\texttt{hops}$, $\texttt{fundee}$, c_{guest}) by $\bar{P}$:

  ensure $State = \text{OPEN}$
  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
  $State \leftarrow$ VIRTUALISING
  $\texttt{locked}_{\bar{P}} \leftarrow \texttt{locked}_{\bar{P}} + c$ // if $\bar{P}$ is hosting the $\texttt{funder}$, $\bar{P}$ 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($\texttt{hops}$) > 1 **then** // we are not the last hop
    define new VIRT ITI $\texttt{host}'_P$
    input (VIRTUALISING, $\texttt{host}'_P$, ($sk'_{P,F}$, $pk'_{P,F}$), $pk'_{\bar{P},F}$, $\texttt{hops}$[1:], $\texttt{fundee}$, c_{guest}, $c_{\bar{P}}$, c_P) to $\texttt{hops}$[1].$\texttt{left}$ and expect reply (VIRTUALISING ACK, $\texttt{host\_sibling}$, $pk_{\text{sib},\bar{P},F}$)
    input (INIT, $\texttt{host}_P$, $\texttt{host}'_{\bar{P}}$, $\texttt{host\_sibling}$, ($sk'_{P,F}$, $pk'_{P,F}$), $pk'_{\bar{P},F}$, $pk_{\text{sib},\bar{P},F}$, ($sk_{P,F}$, $pk_{P,F}$), $pk_{\bar{P},F}$, c_{guest}) to $\texttt{host}'_P$ and expect reply (HOST INIT OK)
  else // we are the last hop
    input (INIT, $\texttt{host}_P$, $\texttt{host}'_{\bar{P}}$, $\texttt{fundee}$=$\texttt{fundee}$, ($sk'_{P,F}$, $pk'_{P,F}$), $pk'_{\bar{P},F}$, ($sk_{P,F}$, $pk_{P,F}$), $pk_{\bar{P},F}$, c_{guest}) to new VIRT ITI $\texttt{host}'_P$ and expect reply (HOST INIT OK)
  end if
  $State \leftarrow$ WAITING FOR REVOCATION
  send (VIRTUALISING ACK, $\texttt{host}'_P$, $pk'_{P,F}$) to $\bar{P}$

On input (VIRTUALISING, $\texttt{host\_sibling}$, ($sk'_{P,F}$, $pk'_{P,F}$), $pk_{\text{sib},\bar{P},F}$, $\texttt{hops}$, $\texttt{fundee}$, c_{guest}, $c_{\text{sib,rem}}$, $_{\text{sib}}$) by $\texttt{sibling}$:

  ensure $State = \text{OPEN}$
  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 stays idle forever
  $State \leftarrow$ VIRTUALISING
  $\texttt{locked}_P \leftarrow \texttt{locked}_P + c$
  define new VIRT ITI $\texttt{host}'_P$
  send (VIRTUALISING, $\texttt{host}'_P$, $pk'_{P,F}$, $\texttt{hops}$, $\texttt{fundee}$, c_{guest}) to $\texttt{hops}$[0].$\texttt{right}$ 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, $\texttt{host}_P$, $\texttt{host}'_{\bar{P}}$, $\texttt{host\_sibling}$, ($sk'_{P,F}$, $pk'_{P,F}$), $pk'_{\bar{P},F}$, $pk_{\text{sib},\bar{P},F}$, ($sk_{P,F}$, $pk_{P,F}$), $pk_{\bar{P},F}$, c_{guest}) to $\texttt{host}'_P$ and expect reply (HOST INIT OK)
  $State \leftarrow$ WAITING FOR REVOCATION
  output (VIRTUALISING ACK, $\texttt{host}'_P$, $pk'_{\bar{P},F}$) to $\texttt{sibling}$

Fig. 17.

Process LN.SIGNATURESROUNDTRIP()

$C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with x coins moved from P's to $\bar{P}$'s output
$\text{sig}_{P,C,i+1} \leftarrow \text{SIGN}(C_{\bar{P},i+1}, sk_{P,F})$ // kept by $\bar{P}$
send (PAY, x, $\text{sig}_{P,C,i+1}$) to $\bar{P}$
// P refers to payer and $\bar{P}$ to payee both in local and remote code
ensure `State` = OPEN
if $\texttt{host}_{\bar{P}} \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$ 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
end if
$C_{\bar{P},i+1} \leftarrow C_{\bar{P},i}$ with x coins moved from P's to $\bar{P}$'s output
ensure $\text{VERIFY}(C_{\bar{P},i+1}, \text{sig}_{P,C,i+1}, pk_{P,F}) = \text{True}$
$C_{P,i+1} \leftarrow C_{P,i}$ with x coins moved from P's to $\bar{P}$'s output
$\text{sig}_{\bar{P},C,i+1} \leftarrow \text{SIGN}(C_{P,i+1}, sk_{\bar{P},F})$ // kept by P
$R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: $(c_{\bar{P}}, pk_{P,\text{out}})$}
$\text{sig}_{\bar{P},R,i} \leftarrow \text{SIGN}(R_{P,i}, sk_{\bar{P},R})$
reply (COMMITMENT SIGNED, $\text{sig}_{\bar{P},C,i+1}$, $\text{sig}_{\bar{P},R,i}$)
$C_{P,i+1} \leftarrow C_{P,i}$ with x coins moved from P's to $\bar{P}$'s output

Fig. 18.

Process LN.REVOCATIONSTRIP()

ensure VERIFY($C_{P,i+1}$, $\text{sig}_{\bar{P},C,i+1}$, $pk_{\bar{P},F}$) = True
$R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: ($c_{\bar{P}}$, $pk_{P,\text{out}}$)}
ensure VERIFY($R_{P,i}$, $\text{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
$R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: (c_P, $pk_{\bar{P},\text{out}}$)}
$\text{sig}_{P,R,i} \leftarrow$ SIGN($R_{\bar{P},i}$, $sk_{P,R}$)
add x to `paid_out`
$c_P \leftarrow c_P - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x;\ i \leftarrow i + 1$
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge$ we have a `host_sibling` **then** // we are intermediary channel
    input (EW 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
send (REVOKE AND ACK, $\text{sig}_{P,R,i}$) to $\bar{P}$
$R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: (c_P, $pk_{\bar{P},\text{out}}$)}
ensure VERIFY($R_{\bar{P},i}$, $\text{sig}_{P,R,i}$, $pk_{P,R}$) = True
add x to `paid_in`
$c_P \leftarrow c_P - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x;\ i \leftarrow i + 1$
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}} \wedge \bar{P}$ has a `host_sibling` **then** // we are intermediary channel
    input (NEW BALANCE, $c_{\bar{P}}$, c_P) to $\texttt{host}_{\bar{P}}$
    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

Fig. 19.

Process LN – On (PAY, x):

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

Fig. 20.

Process LN – On (CHECK FOR LATERAL CLOSE):

if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then**
    input (CHECK FOR LATERAL CLOSE) to $\texttt{host}_P$
end if

Fig. 21.

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

ensure $\mathit{State} \notin \{\bot, \text{INIT}, \text{TOPPED UP}\}$ // channel open
// even virtual channels check $\mathcal{G}_{\text{Ledger}}$ directly. This is intentional
input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to Σ
`last_poll` $\leftarrow |\Sigma|$
if $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$ **then** // counterparty has closed maliciously
    $\mathit{State} \leftarrow \text{CLOSING}$
    LN.SUBMITANDCHECKREVOCATION(j)
    $\mathit{State} \leftarrow \text{CLOSED}$
end if

Fig. 22.

Process $\textsc{ln.submitAndCheckRevocation}(j)$

$\text{sig}_{P,R,j} \leftarrow \textsc{sign}(R_{P,j}, sk_{P,R})$
input ($\textsc{submit}$, $(R_{P,j}, \text{sig}_{P,R,j}, \text{sig}_{\bar{P},R,j})$) to $\mathcal{G}_{\text{Ledger}}$
while $\nexists R_{P,j} \in \Sigma$ **do**
    wait for input ($\textsc{check revocation}$) // ignore other messages
    input ($\textsc{read}$) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
end while
$c_P \leftarrow c_P + c_{\bar{P}}$
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then**
    input ($\textsc{used revocation}$) to $\texttt{host}_P$
end if

Fig. 23.

Process LN – On (CLOSE):

ensure $State \notin \{\perp, \text{INIT}, \text{TOPPED UP}, \text{CLOSED}, \text{BASE PUNISHED}\}$ // channel open
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then** // we have a virtual channel
    $State \leftarrow$ HOST CLOSING
    input (CLOSE) to $\texttt{host}_P$ and keep relaying inputs (CHECK CHAIN FOR CLOSING) to $\texttt{host}_P$ until receiving output (CLOSED) by $\texttt{host}_P$
    $\texttt{host}_P \leftarrow \mathcal{G}_{\text{Ledger}}$
end if
$State \leftarrow$ CLOSING
input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
if $C_{\bar{P},i} \in \Sigma$ **then** // counterparty has closed honestly
    no-op // do nothing
else if $\exists 0 \leq j < i : C_{\bar{P},j} \in \Sigma$ **then** // counterparty has closed maliciously
    LN.SUBMITANDCHECKREVOCATION(j)
else // counterparty is idle
    while $\nexists$ unspent $\texttt{output} \in \Sigma$ that $C_{P,i}$ can spend **do** // possibly due to an active timelock
        wait for input (CHECK VIRTUAL) // ignore other messages
        input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
    end while
    // provably reachable – TODO: ref
    $\text{sig}'_{P,C,i} \leftarrow \text{SIGN}(C_{P,i}, sk_{P,F})$
    input (SUBMIT, $(C_{P,i}, \text{sig}_{P,C,i}, \text{sig}'_{P,C,i})$) to $\mathcal{G}_{\text{Ledger}}$
    while $C_{P,i} \notin \Sigma$ **do**
        wait for input (CHECK CLOSED) // ignore other messages
        input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
    end while
    // provably reachable – TODO: ref
end if
$State \leftarrow$ CLOSED
output (CLOSED)

Fig. 24.

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

$State \leftarrow$ BASE PUNISHED

Fig. 25.

Process VIRT

On every activation, before handling the message:
  if `last_poll` $\neq \bot$ **then** // virtual layer is ready
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    if `last_poll` $+ t < |\Sigma|$ **then**
      for $P \in \{$`guest`, `funder`, `fundee`$\}$ **do** // at most 1 of `funder`, `fundee` is defined
        ensure P.NEGLIGENT() returns (OK)
      end for
    end if
  end if

// `guest` is trusted to give sane inputs, therefore a state machine and input verification is redundant
On input (INIT, $\texttt{host}_P$, $\bar{P}$, `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_{guest}) by `guest`:
  store message contents and `guest` // `sibling`, $pk_{\text{sib},\bar{P},F}$ are missing for edge nodes, `fundee` is present only in last node
  `last_poll` $\leftarrow \bot$
  output (HOST INIT OK) to `guest`

On input (HOST ME, `funder`, `fundee`, $\bar{P}$, $\texttt{host}_P$, c_{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`:
  `last_poll` $\leftarrow \bot$
  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`

Fig. 26.

Process VIRT.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`
    extract $(pk_{\text{left,guest}}, pk_{\text{right,guest}})$ from `left_data`
    output (CHECK KEYS, $(pk_{\text{left,guest}}, pk_{\text{right,guest}})$) to `fundee` and expect reply (KEYS OK)
    return $pk_{\text{loc,virt}}$
  end if
else // we are `host_funder`
  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`
  return (OK)
end if

Fig. 27.

Process VIRT

GETMIDTXS(c_{guest}, c_{loc}, c_{rem}, c_{sib}, c_{sibRem}, $pk_{\text{left,fund}}$, $pk_{\text{loc,fund}}$, $pk_{\text{sib,fund}}$, $pk_{\text{right,fund}}$, $pk_{\text{left,virt}}$, $pk_{\text{loc,virt}}$, $pk_{\text{sib,virt}}$, $pk_{\text{right,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, $pk_{\text{loc,out}}$, $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$):

  ensure $c_{\text{sibRem}} \geq c_{\text{guest}} \land c_{\text{loc}} \geq c_{\text{guest}}$

  $c_{\text{left}} \leftarrow c_{\text{sib}} + c_{\text{sibRem}}$; $c_{\text{right}} \leftarrow c_{\text{loc}} + c_{\text{rem}}$

  $\texttt{left\_fund} \leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{loc,fund}}\}$

  $\texttt{right\_fund} \leftarrow 2/\{pk_{\text{sib,fund}}, pk_{\text{right,fund}}\}$

  $\texttt{left\_virt} \leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}\}$

  $\texttt{left\_virt\_checked} \leftarrow 4/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  $\texttt{right\_virt} \leftarrow 2/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}\}$

  $\texttt{right\_virt\_checked} \leftarrow 4/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  $\texttt{left\_out\_checked} \leftarrow (\texttt{2} \land \texttt{left\_virt\_checked}) \lor (\texttt{3} \land \texttt{left\_virt} + (t+s))$

  $\texttt{right\_out} \leftarrow (\texttt{1} \land \texttt{right\_virt}) \lor (\texttt{3} \land \texttt{right\_virt} + (t+s))$

  $\texttt{right\_out\_checked} \leftarrow (\texttt{1} \land \texttt{right\_virt\_checked}) \lor (\texttt{3} \land \texttt{right\_virt} + (t+s))$

  $\texttt{guest\_all} \leftarrow \texttt{5} \land n/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1 \dots n}\}\}$

  $\texttt{guest\_out} \leftarrow \texttt{4} \land 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  $\texttt{guest} \leftarrow (\texttt{guest\_out} + (t+s)) \lor \texttt{guest\_all}$

  $\text{TX}_{\text{none}} \leftarrow$ TX {inputs: ((c_{left}, $\texttt{left\_fund}$), (c_{right}, $\texttt{right\_fund}$)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, $\texttt{left\_out\_checked}$), ($c_{\text{right}} - c_{\text{guest}}$, $\texttt{right\_out\_checked}$), ($c_{\text{guest}}$, $pk_{\text{loc,out}}$), (c_{guest}, $\texttt{guest}$))}

  $\text{TX}_{\text{left}} \leftarrow$ TX {inputs: (($c_{\text{left}} - c_{\text{guest}}$, $\texttt{1} \land \texttt{left\_virt\_checked}$), ($c_{\text{right}}$, $\texttt{right\_fund}$)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, $\texttt{3} \land \texttt{left\_virt}$), ($c_{\text{right}} - c_{\text{guest}}$, $\texttt{right\_out\_checked}$), ($c_{\text{guest}}$, $pk_{\text{loc,out}}$))}

  $\text{TX}_{\text{right}} \leftarrow$ TX {inputs: ((c_{left}, $\texttt{left\_fund}$), ($c_{\text{right}} - c_{\text{guest}}$, $\texttt{2} \land \texttt{right\_virt\_checked}$), ($c_{\text{guest}}$, $\texttt{guest\_all}$)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, $\texttt{left\_out\_checked}$), ($c_{\text{right}} - c_{\text{guest}}$, $\texttt{3} \land \texttt{right\_virt}$), ($c_{\text{guest}}$, $pk_{\text{loc,out}}$), (c_{guest}, $\texttt{guest}$))}

  $\text{TX}_{\text{both}} \leftarrow$ TX {inputs: (($c_{\text{left}} - c_{\text{guest}}$, $\texttt{1} \land \texttt{left\_virt\_checked}$), ($c_{\text{right}} - c_{\text{guest}}$, $\texttt{2} \land \texttt{right\_virt\_checked}$), ($c_{\text{guest}}$, $\texttt{guest\_all}$)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, $\texttt{3} \land \texttt{left\_virt}$), ($c_{\text{right}} - c_{\text{guest}}$, $\texttt{3} \land \texttt{right\_virt}$), ($c_{\text{guest}}, pk_{\text{loc,out}}$))}

  return (TX_{none}, TX_{left}, TX_{right}, TX_{both})

Fig. 28.

Process virt

// 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.

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`):

  ensure $c_{\text{left}} \geq c_{\text{guest}}$
  $c_{\text{tot}} \leftarrow c_{\text{left}} + c_{\text{right}}$
  `fund` $\leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{right,fund}}\}$
  `virt` $\leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}\}$
  `virt_checked` $\leftarrow 4/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
  if `is_funder` $=$ True **then**
    `out` $\leftarrow$ (`1` $\wedge$ `virt_checked`) $\vee$ (`3` $\wedge$ `virt` $+ (t + s)$)
  else // TXs belong to `fundee`
    `out` $\leftarrow$ (`2` $\wedge$ `virt_checked`) $\vee$ (`3` $\wedge$ `virt` $+ (t + s)$)
  end if
  `guest_all` $\leftarrow$ `5` $\wedge\, n/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}, \{pk_{\text{sec},1 \ldots n}\}\}$
  `guest_out` $\leftarrow$ `4` $\wedge\, 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
  `guest` $\leftarrow$ (`guest_out` $+ (t + s)$) $\vee$ `guest_all`
  $\text{TX}_{\text{base}} \leftarrow$ TX {input: (c_{tot}, `fund`), outputs: (($c_{\text{tot}} - c_{\text{guest}}$, `out`), ($c_{\text{guest}}$, `guest`))}
  return TX_{base}

Fig. 29.

Process $\textsc{virt.siblingSigs}()$

parse input as $\text{sigs}_{\text{byLeft}}$
$(\text{TX}_{\text{loc,none}}, \text{TX}_{\text{loc,left}}, \text{TX}_{\text{loc,right}}, \text{TX}_{\text{loc,both}}) \leftarrow \textsc{virt.getMidTXs}(c_{\text{guest}}, c_P, c_{\bar{P}}, c_{\text{sib}}, c_{\text{sib,rem}}, pk_{\text{sib,rem},F}, pk_{\text{sib},F}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{sib,rem,virt}}, pk_{\text{loc,virt}}, pk_{\text{loc,virt}}, pk_{\text{rem,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}, pk_{\text{loc,virt}}, \{pk_{\text{sec},i}\}_{i \in 1 \dots n})$
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 ($i \in [2, \dots, n-1] \setminus \{j\}$) **do** // j is our hop number, hop data encoded in `left_data` and `right_data`
  extract data needed for $\textsc{getMidTXs}()$ from `left_data` (if $i < j$) or `right_data` (if $i > j$) and assign it to $\texttt{data}_i$ and $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$ // P and `comm_keys` are missing, that is OK. $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$ contains each party's $pk_{i,\text{virt}}$
  $(\text{TX}_{i,\text{none}}, \text{TX}_{i,\text{left}}, \text{TX}_{i,\text{right}}, \text{TX}_{i,\text{both}}) \leftarrow \textsc{virt.getMidTXs}(\texttt{data}_i, \{pk_{\text{sec},i}\}_{i \in 1 \dots n})$
  add $\textsc{sign}(\text{TX}_{i,\text{right}}, sk_{\text{loc,virt}}, \textsc{anyprevout})$ and $\textsc{sign}(\text{TX}_{i,\text{both}}, sk_{\text{loc,virt}}, \textsc{anyprevout})$ 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 $\textsc{sign}()$ 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 $\textsc{sign}(\text{TX}_{i,\text{left}}, sk_{\text{loc,virt}}, \textsc{anyprevout})$ to $\text{sigs}_{\text{toRight}}$
  else if $j - i = 1$ **then** // hop is our previous
    add $\textsc{sign}(\text{TX}_{i,\text{left}}, sk_{\text{loc,virt}}, \textsc{anyprevout})$ 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`
  $\text{TX}_{\text{next,none}} \leftarrow \textsc{virt.getEdgeTXs}(c_{\text{guest}}, c_P, c_{\bar{P}}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{loc,virt}}, pk_{\text{rem,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}, \text{False})$
end if
call $\bar{P}.\textsc{circulateVirtualSigs}(\text{sigs}_{\text{toRight}})$ 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 $(\textsc{virtualSigsBack}, \text{sigs}_{\text{toLeft}}, \text{sigs}_{\text{byRight}})$

Fig. 30.

Process $\textsc{virt.intermediarySigs}()$

$(\mathrm{TX}_{\mathrm{loc,none}}, \mathrm{TX}_{\mathrm{loc,left}}, \mathrm{TX}_{\mathrm{loc,right}}, \mathrm{TX}_{\mathrm{loc,both}}) \leftarrow \textsc{virt.getMidTXs}(c_{\mathrm{guest}}, c_P, c_{\bar{P}}, c_{\mathrm{sib}}, c_{\mathrm{sib,rem}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{sib},F}, pk_{\mathrm{sib,rem}F}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{sib,rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, pk_{\mathrm{loc,virt}}, \{pk_{\mathrm{sec},i}\}_{i\in 1\dots n})$

// not verifying our signatures in $\mathrm{sigs}_{\mathrm{byLeft}}$, our (trusted) `sibling` will do that

input ($\textsc{virtual sigs forward}$, $\mathrm{sigs}_{\mathrm{byLeft}}$) to `sibling`

$\textsc{virt.siblingSigs}()$

$\mathrm{sigs}_{\mathrm{toLeft}} \leftarrow \mathrm{sigs}_{\mathrm{byRight}} + \mathrm{sigs}_{\mathrm{toLeft}}$

if `left_data` does not contain data from a second-previous hop **then** // previous hop is `host_funder`

    $\mathrm{TX}_{\mathrm{prev,none}} \leftarrow \textsc{virt.getEdgeTXs}(c_{\mathrm{guest}}, c_{\bar{P}}, c_P, pk_{\mathrm{rem},F}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, \mathrm{True})$

end if

return $\mathrm{sigs}_{\mathrm{toLeft}}$

Fig. 31.

Process $\textsc{virt.hostFundeeSigs}()$

$\mathrm{TX}_{\mathrm{loc,none}} \leftarrow \textsc{virt.getEdgeTXs}(c_{\mathrm{guest}}, c_P, c_{\bar{P}}, pk_{\mathrm{loc},F}, pk_{\mathrm{rem},F}, pk_{\mathrm{loc,virt}}, pk_{\mathrm{rem,virt}}, pk_{\mathrm{left,guest}}, pk_{\mathrm{right,guest}}, \mathrm{False})$

for each hop apart from the first and ours ($i \in [2, \dots, n-1]$) **do** // hop data encoded in `left_data`

    extract data needed for $\textsc{getMidTXs}()$ from `left_data` and assign it to $\mathtt{data}_i$ and $\{pk_{\mathrm{sec},i}\}_{i\in 1\dots n}$ // $\{pk_{\mathrm{sec},i}\}_{i\in 1\dots n}$ contains each party's $pk_{i,\mathrm{virt}}$

    $(\mathrm{TX}_{i,\mathrm{none}}, \mathrm{TX}_{i,\mathrm{left}}, \mathrm{TX}_{i,\mathrm{right}}, \mathrm{TX}_{i,\mathrm{both}}) \leftarrow \textsc{virt.getMidTXs}(\mathtt{data}_i, \{pk_{\mathrm{sec},i}\}_{i\in 1\dots n})$

    add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{right}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ and $\textsc{sign}(\mathrm{TX}_{i,\mathrm{both}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$ // if i-th hop is adjacent, 2 signatures will be produced by each $\textsc{sign}()$ invocation: one for the `guest_all` and one for the $2 \wedge$ `right_virt_checked` input

    output ($\textsc{sign TXs}$, $\mathrm{TX}_{i,\mathrm{left}}$, $\mathrm{TX}_{i,\mathrm{right}}$, $\mathrm{TX}_{i,\mathrm{both}}$) to `fundee` and expect reply ($\textsc{TXs signed}$, $\mathrm{sigs}_{\mathrm{guest}}$)

    add $\mathrm{sigs}_{\mathrm{guest}}$ to $\mathrm{sigs}_{\mathrm{toLeft}}$

    if $i = n - 1$ **then** // hop is our previous

        add $\textsc{sign}(\mathrm{TX}_{i,\mathrm{left}}, sk_{\mathrm{loc,virt}}, \textsc{anyprevout})$ to $\mathrm{sigs}_{\mathrm{toLeft}}$

    end if

end for

return $\mathrm{sigs}_{\mathrm{toLeft}}$

Fig. 32.

Process virt.hostFunderSigs()

for each hop apart from the last and ours ($i \in [2, \dots, n-1]$) **do** // hop data encoded in `right_data`
    extract data needed for getMidTXs() from `right_data` and assign it to $\texttt{data}_i$ and $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$ // $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$ 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($\texttt{data}_i$, $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$)
    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
    output (sign TXs, $\text{TX}_{i,\text{left}}$, $\text{TX}_{i,\text{right}}$, $\text{TX}_{i,\text{both}}$) to `fundee` and expect reply (TXs signed, $\text{sigs}_{\text{guest}}$)
    add $\text{sigs}_{\text{guest}}$ to $\text{sigs}_{\text{toRight}}$
    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)

Fig. 33.

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`
        return virt.hostFundeeSigs()
    end if
else // we are `host_funder`
    return virt.hostFunderSigs()
end if

Fig. 34.

Process $\textsc{virt.circulateFundingSigs}(\text{sig}_{\text{loc,none}})$

if $\text{sig}_{\text{loc,none}}$ is given as argument **then** // we are not `host_funder`
    ensure $\textsc{verify}(\text{TX}_{\text{loc,none}}, \text{sig}_{\text{loc,none}}, pk_{\text{prev},F}) = \text{True}$ // $pk_{\text{prev},F}$, found in `left_data`
    $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
    if we have a `sibling` **then** // we are not `host_fundee`
        input $(\textsc{virtual base sig forward}, \text{sig}_{\text{loc,none}})$ to `sibling` // `sibling` needs $\text{sig}_{\text{loc,none}}$ for closing
        $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
        $\text{sig}_{\text{next,none}} \leftarrow \textsc{sign}(\text{TX}_{\text{next,none}}, sk_{\text{loc},F})$
        call $\textsc{virt.circulateVirtualSigs}(\text{sig}_{\text{next,none}})$ of $\bar{P}$ and assign returned value to $\text{sig}_{\text{loc,none}}$
        ensure $\textsc{verify}(\text{TX}_{\text{loc,none}}, \text{sig}_{\text{loc,none}}, pk_{\text{next},F}) = \text{True}$ // $pk_{\text{next},F}$, found in `right_data`
        add $\text{sig}_{\text{loc,none}}$ to $\text{sigs}_{\text{loc,none}}$
        output $(\textsc{virtual base sig back}, \text{sig}_{\text{loc,none}})$ // sibling needs $\text{sig}_{\text{loc,none}}$ for closing
        add $\text{sig}_{\text{loc,none}}$ to $\text{sigs}_{\text{loc,none}}$
    end if
    $\text{sig}_{\text{prev,none}} \leftarrow \textsc{sign}(\text{TX}_{\text{prev,none}}, sk_{\text{loc},F})$
    return $\text{sig}_{\text{prev,none}}$
else // we are `host_funder`
    $\text{sig}_{\text{next,none}} \leftarrow \textsc{sign}(\text{TX}_{\text{next,none}}, sk_{\text{loc},F})$
    call $\textsc{virt.circulateFundingSigs}(\text{sig}_{\text{next,none}})$ of $\bar{P}$ and assign returned value to $\text{sig}_{\text{loc,none}}$
    ensure $\textsc{verify}(\text{TX}_{\text{loc,none}}, \text{sig}_{\text{loc,none}}, pk_{\text{next},F}) = \text{True}$ // $pk_{\text{next},F}$ found in `right_data`
    $\text{sigs}_{\text{loc,none}} \leftarrow \{\text{sig}_{\text{loc,none}}\}$
    return $(\textsc{ok})$
end if

Fig. 35.

Process VIRT.CIRCULATEREVOCATIONS(revoc_by_prev)

if revoc_by_prev is given as argument **then** // we are not host_funder
    ensure guest.PROCESSREMOTEREVOCATION(revoc_by_prev) returns (OK)
else // we are host_funder
    revoc_for_next ← guest.REVOKEPREVIOUS()
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    last_poll ← $|\Sigma|$
    call VIRT.CIRCULATEREVOCATIONS(revoc_for_next) of $\bar{P}$ and assign returned value to revoc_by_next
    ensure guest.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK) // If the "ensure" fails, the opening process freezes, this is intentional. The channel can still close via (CLOSE)
    return (OK)
end if
if we have a sibling **then** // we are not host_fundee nor host_funder
    input (VIRTUAL REVOCATION FORWARD) to sibling
    revoc_for_next ← guest.REVOKEPREVIOUS()
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign ouput to Σ
    last_poll ← $|\Sigma|$
    call VIRT.CIRCULATEREVOCATIONS(revoc_for_next) of $\bar{P}$ and assign output to revoc_by_next
    ensure guest.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK)
    output (VIRTUAL REVOCATION BACK)
end if
revoc_for_prev ← guest.REVOKEPREVIOUS()
output (HOSTS READY) to guest and expect reply (HOST ACK)
return revoc_for_prev // we are not host_fundee nor host_funder

Fig. 36.

Process VIRT – poll

On input (CHECK FOR LATERAL CLOSE) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$:
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
    $\texttt{prev\_went\_on\_chain} \leftarrow \text{TX}_{\text{prev,left}} \in \Sigma \vee \text{TX}_{\text{prev,none}} \in \Sigma$
    $\texttt{next\_went\_on\_chain} \leftarrow \text{TX}_{\text{next,right}} \in \Sigma \vee \text{TX}_{\text{next,none}} \in \Sigma$
    $\texttt{last\_poll} \leftarrow |\Sigma|$
    if $\texttt{prev\_went\_on\_chain} \vee \texttt{next\_went\_on\_chain}$ **then**
        ignore all messages except for (CHECK CHAIN FOR CLOSING) by R
        $State \leftarrow$ CLOSING
    end if
    if $\texttt{prev\_went\_on\_chain} \wedge \texttt{next\_went\_on\_chain}$ **then**
        VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,both}}$, $\text{sigs}_{\text{loc,both}}$)
    else if $\texttt{prev\_went\_on\_chain}$ **then**
        VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,left}}$, $\text{sigs}_{\text{loc,left}}$)
    else if $\texttt{next\_went\_on\_chain}$ **then**
        VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,right}}$, $\text{sigs}_{\text{loc,right}}$)
    end if

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

Fig. 37.

Process VIRT – close

On input (CLOSE) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$: // At most one of `funder`, `fundee` is defined
  if *State* = CLOSED **then** output (CLOSED) to R
  if *State* = GUEST PUNISHED **then** output (GUEST PUNISHED) to R
  ensure *State* = OPEN
  if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then** // $\texttt{host}_P$ is a VIRT
    ignore all messages except for output (CLOSED) by $\texttt{host}_P$. Also relay to $\texttt{host}_P$ any (CHECK CHAIN FOR CLOSING) input received
    input (CLOSE) to $\texttt{host}_P$
  end if
  // if we have a $\texttt{host}_P$, continue from here on output (CLOSED) by it
  send (READ) to $\mathcal{G}_{\text{Ledger}}$ as R and assign reply to Σ
  let `tx` be the unique valid TX for Σ 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 R
  State ← CLOSING
  send (SUBMIT, (`tx`, `sigs`)) to $\mathcal{G}_{\text{Ledger}}$

On (CHECK CHAIN FOR CLOSING) by $R \in \{\texttt{guest}, \texttt{funder}, \texttt{fundee}\}$:
  ensure *State* = CLOSING
  send (READ) to $\mathcal{G}_{\text{Ledger}}$ as R and assign reply to Σ
  if $R = \texttt{guest}$ **then**
    $pk_1 \leftarrow pk_{\text{left,guest}}$; $pk_2 \leftarrow pk_{\text{right,guest}}$
  else // $R \in \{\texttt{funder}, \texttt{fundee}\}$
    $pk_1 \leftarrow pk_{\text{loc,virt}}$; $pk_2 \leftarrow pk_{\text{rem,virt}}$
  end if
  if Σ 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
    State ← CLOSED
    output (CLOSED) to R
  end if

On input (USED REVOCATION) by `guest`: // (USED REVOCATION) by `funder`/`fundee` is ignored
  State ← GUEST PUNISHED
  input (USED REVOCATION) to $\texttt{host}_P$, expect reply (USED REVOCATION OK)
  if `funder` it `fundee` is defined **then** output USED REVOCATION to it

On output (USED REVOCATION) by $\texttt{host}_P$:
  State ← GUEST PUNISHED
  if `funder` it `fundee` is defined **then** output USED REVOCATION to it

Fig. 38.

Lemma 1 (Real world balance security). *Consider a real world execution with $P \in \{Alice, Bob\}$ honest* LN *ITI and $\bar{P}$ the counterparty ITI. Assume that all of the following are true:*

- *the internal variable* `negligent` *of P has value "False",*
- *P has transitioned to the* OPEN *State for the first time after having received $(\text{OPEN}, c, \dots)$ by either $\mathcal{E}$ or $\bar{P}$,*
- *P [has received $(\text{FUND ME}, f_i, \dots)$ as input by another* LN *ITI while State was* OPEN *and subsequently P transitioned to* OPEN *State] n times,*
- *P [has received (PAY, d_i) by $\mathcal{E}$ while State was* OPEN *and P subsequently transitioned to* OPEN *State] m times,*
- *P [has received $(\text{PAY}, e_i, \dots)$ by $\bar{P}$ while State was* OPEN *and P subsequently transitioned to* OPEN *State] l times.*

Let $\phi = 1$ if $P = Alice$, or $\phi = 0$ if $P = Bob$. If P receives (CLOSE) *by $\mathcal{E}$ and, if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ the output of $\texttt{host}_P$ is* (CLOSED)*, then eventually the state obtained when P inputs* (READ) *to $\mathcal{G}_{\text{Ledger}}$ will contain h $(c_i, pk_{P,\text{out}})$ outputs such that*

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

with overwhelming probability in the security parameter.

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

- has received (OPEN, c, …) while in the INIT *State* by either $\mathcal{E}$ if ($P = Alice$, Fig. 14, l. 1) or $\bar{P}$ (if $P = Bob$, Fig. 3, l. 3),
- it internally holds a signature on the commitment transaction $C_{P,0}$ that is valid by $pk_{\bar{P},F}$ (Fig. 11, ll. 12 and 23),
- it either has the transaction F in the $\mathcal{G}_{\text{Ledger}}$ state if $\texttt{host} = \mathcal{G}_{\text{Ledger}}$ (Fig. 12, l. 3), or is able to cause an eventual inclusion of a $(c, 2/\{pk_{A,F}, pk_{B,F}\})$ output in the $\mathcal{G}_{\text{Ledger}}$ state by passing input (CLOSE) to `host`. The latter is true if `hops`[0].`left` (when $P = Alice$, Fig. 10, l. 6) or `hops`[-1].`right` (when $P = Bob$, Fig. 13, l. 8) has passed output (FUNDED, …) to P, as the outputting party is trusted (as it executes locally) and has itself been passed output (HOSTS READY) by its (also trusted) `host` VIRT ITI (Fig. 16, l. 12). The `host` only outputs (HOSTS READY) when its `guest` (which is the aforementioned party that output (FUNDED, …)) verifies that the remote revocation has happened correctly (Fig. 36, ll. 2 or 8). As we will see later, this verification ensures that either the aforementioned output will end up in the $\mathcal{G}_{\text{Ledger}}$ state, or the `guest`'s counterparty will lose all their funds to the `guest`.

□

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

- *the internal variable* $\texttt{ignore}_P$ *of* $\mathcal{F}_{\text{Chan}}$ *has the value "False",*
- $\mathcal{F}_{\text{Chan}}$ *has received* $(\textsc{open}, c)$ *by* $\mathcal{S}$*,*
- $\mathcal{F}_{\text{Chan}}$ *has received* $(\textsc{fund}, f_i, P)$ *by* $\mathcal{S}$ *while State was* OPEN $n \geq 0$ *times,*
- $\mathcal{F}_{\text{Chan}}$ *has received* $(\textsc{pay}, d_i, P, \bar{P})$ *by* $\mathcal{S}$ *while State was* OPEN $m \geq 0$ *times,*
- $\mathcal{F}_{\text{Chan}}$ *has received* $(\textsc{pay}, e_i, \bar{P}, P)$ *by* $\mathcal{S}$ *while State was* OPEN $l \geq 0$ *times.*

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

$$\texttt{balance}_P = \phi \cdot c - \sum_{i=1}^{n} f_i - \sum_{i=1}^{m} d_i + \sum_{i=1}^{l} e_i \tag{2}$$

Proof. We will prove the Lemma by following the evolution of the $\texttt{balance}_P$ variable.

- When $\mathcal{F}_{\text{Chan}}$ is initialised, it sets $\texttt{balance}_A \leftarrow 0$ and $\texttt{balance}_B \leftarrow 0$ (Fig. **??**, l. 1).
- When $\mathcal{F}_{\text{Chan}}$ receives $(\textsc{open}, c)$ by $\mathcal{S}$ when the former is in the INIT *State*, it transitions away from this state for the remainder of the execution and sets $\texttt{balance}_A \leftarrow c$ (Fig. **??**, l. **??**). This happens at most once during the entire execution and is necessary to transition to the OPEN *State*.
- Every time $\mathcal{F}_{\text{Chan}}$ receives $(\textsc{fund}, f_i, P)$ by $\mathcal{S}$ while the former is in the OPEN *State*, it decrements $\texttt{balance}_P$ by f_i (Fig. **??**, l. 1). If $(\textsc{fund}, f_i, P)$ is received $n \geq 0$ times, $\texttt{balance}_P$ will be decremented by $\sum\limits_{i=1}^{n} f_i$ in total.
- Every time $\mathcal{F}_{\text{Chan}}$ receives $(\textsc{pay}, d_i, P, \bar{P})$ by $\mathcal{S}$ while the former is in the OPEN *State*, it decrements $\texttt{balance}_P$ by d_i (Fig. **??**, l. **??**). If $(\textsc{pay}, d_i, P, \bar{P})$ is received $m \geq 0$ times, $\texttt{balance}_P$ will be decremented by $\sum\limits_{i=1}^{m} d_i$ in total.
- Every time $\mathcal{F}_{\text{Chan}}$ receives $(\textsc{pay}, e_i, \bar{P}, P)$ by $\mathcal{S}$ while the former is in the OPEN *State*, it increments $\texttt{balance}_P$ by e_i (Fig. **??**, l. **??**). If $(\textsc{pay}, e_i, \bar{P}, P)$ is received $l \geq 0$ times, $\texttt{balance}_P$ will be incremented by $\sum\limits_{i=1}^{l} e_i$ in total.

On aggregate, after the above are completed and when $\mathcal{F}_{\text{Chan}}$ receives (CLOSE, P) or (CLOSE, $\bar{P}$) by $\mathcal{S}$ and assuming $\texttt{ignore}_P = \text{False}$, $\texttt{balance}_P = c - \sum\limits_{i=1}^{n} f_i - \sum\limits_{i=1}^{m} d_i + \sum\limits_{i=1}^{l} e_i$ if $P = Alice$, or else if $P = Bob$, $\texttt{balance}_P = -\sum\limits_{i=1}^{n} f_i - \sum\limits_{i=1}^{m} d_i + \sum\limits_{i=1}^{l} e_i$. □

Lemma 3 (No halt). *In an ideal execution with* $\mathcal{F}_{\text{Chan}}$ *and* $\mathcal{S}$*, the functionality halts with negligible probability in the security parameter (i.e. l.* **??** *of Fig.* **??** *is executed negligibly often).*

Proof. The only way for $\mathcal{F}_{\text{Chan}}$ to halt is if either $\texttt{check}_A$ or $\texttt{check}_B$ fails. For these checks to take place, $\mathcal{F}_{\text{Chan}}$ must have received (CLOSE, *Alice*) or (CLOSE, *Bob*) by $\mathcal{S}$ while in the OPEN *State* (Fig. **??**, l. **??**). Additionally, $\mathcal{F}_{\text{Chan}}$ can only reach the OPEN *State* if all honest simulated parties transition to the OPEN *State* as well (Fig. 6, l. 10), which in turn, if simulated *Alice* is honest, happens only if she has received (OPEN, ...) by $\mathcal{E}$ (Fig. 14). Observe further that $\mathcal{S}$ notifies $\mathcal{F}_{\text{Chan}}$ right away when either simulated party becomes corrupted or negligent (Fig. 6, ll. 3 and 5 respectively) and the balance of each party is checked by $\mathcal{F}_{\text{Chan}}$ only if this party is not corrupted nor negligent. What is more, $\mathcal{S}$ informs $\mathcal{F}_{\text{Chan}}$ of all successful FUNDs and PAYs (Fig. 6, ll. 13 and 11 respectively). These facts in combination mean that for each party, whenever the prerequisites for Lemma 2 are true the prerequisites for Lemma 1 are also true (with the same n, m, l values) and therefore the check for this party will succeed (Fig **??**, l. **??**). Furthermore, when the first prerequisite for Lemma 2 does not hold (i.e. $\texttt{ignore}_P$ = False), the check for the respective party will be true, when the second prerequisite does not hold (i.e. *State* $\neq$ OPEN when (CLOSE, *Alice*/*Bob*) is received) the check will not take place at all and the other three prerequisites hold always, therefore the check cannot fail if the prerequisites for Lemma 2 do not hold. On aggregate, when an ideal execution of $\mathcal{F}_{\text{Chan}}$ and $\mathcal{S}$ take place, $\mathcal{F}_{\text{Chan}}$ will halt with negligible probability in the security parameter. □

Theorem 1 (Recursive Virtual Payment Channel Security). *The protocol LN realises $\mathcal{F}_{\text{Chan}}$ with simulator $\mathcal{S}$ given a global functionality $\mathcal{G}_{\text{Ledger}}$ and assuming the security of the underlying digital signature. Specifically,*

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

Proof. By inspection of Figs. 1 and 5 we can deduce that for a particular $\mathcal{E}$, in the ideal world execution $\textsc{exec}^{\mathcal{F}_{\text{Chan}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S}_{\mathcal{A}},\mathcal{E}}$, $\mathcal{S}_{\mathcal{A}}$ simulates internally the two LN parties exactly as they would execute in the real world execution, $\textsc{exec}^{\mathcal{G}_{\text{Ledger}}}_{\text{LN},\mathcal{A},\mathcal{E}}$ in case $\mathcal{F}_{\text{Chan}}$ does not halt. Indeed, $\mathcal{F}_{\text{Chan}}$ only halts with negligible probability according to Lemma 3, therefore the two executions are computationally indistinguishable. □

References