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

Fig. 1.

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

On initalisation:
  $pk_A \leftarrow \perp$; $pk_B \leftarrow \perp$
  $\texttt{balance}_A \leftarrow 0$; $\texttt{balance}_B \leftarrow 0$
  $\texttt{is\_corrupted\_or\_negligent}_A \leftarrow$ False; $\texttt{is\_corrupted\_or\_negligent}_B \leftarrow$ False;
  $\mathit{State} \leftarrow \perp$

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

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

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

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

On (OPEN) by $\mathcal{A}$:
  ensure $\mathit{State} =$ TENTATIVE-OPEN
  $\mathit{State} \leftarrow$ OPEN

On (PAY, x) to P by $\mathcal{E}$:
  ensure $\mathit{State} =$ OPEN
  $c \leftarrow x$
  if $P = \mathit{Alice}$ **then** $S \leftarrow \mathit{Alice}$; $\bar{S} \leftarrow \mathit{Bob}$ **else** $S \leftarrow \mathit{Bob}$; $\bar{S} \leftarrow \mathit{Alice}$
  $\mathit{State} \leftarrow$ PAYING

On (PAY OK) by $\mathcal{A}$:
  ensure $\mathit{State} =$ PAYING
  $\texttt{balance}_S \leftarrow \texttt{balance}_S - c$; $\texttt{balance}_{\bar{S}} \leftarrow \texttt{balance}_{\bar{S}} + c$
  forget $c, S, \bar{S}$
  $\mathit{State} \leftarrow$ OPEN

Fig. 2.

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

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

On (FUND, x) to P by R:
    ensure R is an ITI that runs $\mathcal{F}_{\text{Chan}}$ or Π_{Chan} code
    ensure $\mathit{State} = \text{OPEN}$
    $c \leftarrow x$
    if $P = \mathit{Alice}$ **then** $S \leftarrow \mathit{Alice}$ **else** $S \leftarrow \mathit{Bob}$
    $\mathit{State} \leftarrow \text{FUNDING}$

On (FUND OK) by $\mathcal{A}$:
    ensure $\mathit{State} = \text{FUNDING}$
    $\texttt{balance}_S \leftarrow \texttt{balance}_S - c$
    forget c, S
    $\mathit{State} \leftarrow \text{OPEN}$

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

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

On (CLOSE) by $\mathcal{A}$:
    ensure $\mathit{State} \in \{\text{OPEN}, \text{CLOSING}\}$
    for $P \in \{A, B\}$ **do**
        if $\texttt{is\_corrupted\_or\_negligent}_P = \text{False}$ **then**
            input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign ouput to Σ_P
            $\texttt{coins}_P \leftarrow$ sum of coins exclusively spendable by pk_P in Σ_P
        end if
    end for
    if $(\texttt{is\_corrupted\_or\_negligent}_A \vee \texttt{coins}_A \geq \texttt{balance}_A)$ $\wedge(\texttt{is\_corrupted\_or\_negligent}_B \vee \texttt{coins}_B \geq \texttt{balance}_B)$ **then**
        $\mathit{State} \leftarrow \text{CLOSED}$
    else // balance security is broken
        halt
    end if

Fig. 3.

Simulator $\mathcal{S}$

- 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.

Simulator $\mathcal{S}$– Pt. 1

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

On ($\textsc{open}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}}, pk_{B,F}$, *Bob*) by $\mathcal{F}_{\text{Chan}}$: // *Alice* corrupted
    send LN message ($\textsc{open}$, $pk_{B,F}$) 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 ($\textsc{pay}$, x, *Dave*) by $\mathcal{F}_{\text{Chan}}$:
    if both channel parties are honest **then**
        simulate *Dave* receiving input ($\textsc{pay}$, x) by $\mathcal{E}$
        ensure simulated *Dave* outputs ($\textsc{ok}$)
        send ($\textsc{ok}$) to $\mathcal{F}_{\text{Chan}}$
    else if only *Dave*'s counterparty is corrupted **then** // else just relay to $\mathcal{A}$
        simulate *Dave* receiving input ($\textsc{pay}$, x) by $\mathcal{E}$
        ensure simulated *Dave* outputs ($\textsc{ok}$)
        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 ($\textsc{fund you}$, c, *Bob*, *Charlie*, *Alice*) by $\mathcal{F}_{\text{Chan}}$:
    simulate *Alice* receiving input ($\textsc{fund you}$, c, *Bob*) by *Charlie*
    ensure simulated *Alice* outputs ($\textsc{ok}$) to *Charlie*
    send ($\textsc{ok}$) to $\mathcal{F}_{\text{Chan}}$

On ($\textsc{fund}$ 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 ($\textsc{ok}$) to $\mathcal{F}_{\text{Chan}}$

On ($\textsc{allow fund}$, 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 ($\textsc{ok}$) to the simulated *Alice*
    intercept this message and send it to $\mathcal{F}_{\text{Chan}}$'s *Alice*

Fig. 4.

Simulator $\mathcal{S}$– Pt. 2

On (IS OPEN SUCCESSFUL, id) by $\mathcal{F}_{\text{Chan}}$:

  retrieve and remove from `virtual_opening` the data marked with id
  simulate line **??** of Fig. **??** with *Alice* using this data
  ensure *Alice* completes execution of VChan() successfully
  send (OK) to $\mathcal{F}_{\text{Chan}}$

On (UPDATE TO VIRTUAL) by $\mathcal{F}_{\text{Chan}}$:

  retrieve and remove from `virtual_opening` the data marked with id
  simulate line **??** of Fig. **??** with *Alice* using this data
  ensure *Alice* completes execution of VChan() successfully
  extract from *Alice*'s state the new virtual funding TX V for pre-existing channel
  extract from *Alice*'s state the new commitment TX C that spends the on-chain funding TX
  send (V, C) to $\mathcal{F}_{\text{Chan}}$

On (FUND DONE, id) 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. 5.

Process LN − init

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

On (TOP UP):
    ensure $P = Alice$ // activated party is the funder
    ensure $State^P =$ INIT
    $(sk_{P,\text{chain}}, pk_{P,\text{chain}}) \leftarrow$ KEYGEN()
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P 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 $\mathcal{G}_{\text{Ledger}}$ as P and assign ouput to Σ
    end while
    $State^P \leftarrow$ 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. 6.

Process LN – methods used by VIRT

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

PROCESSREMOTEREVOCATION($\text{sig}_{\bar{P},R,i}$):
  ensure $State^P =$ 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, $pk_{P,\text{out}}$)}
  ensure VERIFY($R_{P,i}$, $\text{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
  if $State^P =$ WAITING FOR REVOCATION **then**
    $State^P \leftarrow$ WAITING FOR OUTBOUND REVOCATION
  else // $State^P =$ WAITING FOR INBOUND REVOCATION
    $i \leftarrow i + 1$
    $State^P \leftarrow$ WAITING FOR HOSTS READY
  end if
  return (OK)

Fig. 7.

Process LN.EXCHANGEOPENKEYS()

$(sk_{A,F}, pk_{A,F}) \leftarrow$ KEYGEN(); $(sk_{A,R}, pk_{A,R}) \leftarrow$ KEYGEN()
if ideal world **then**
    ensure $State^B =$ INIT
    send (OPEN CHANNEL, c, `hops`, $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$) to $\mathcal{A}$ and expect reply (OPEN CHANNEL OK)
    $(sk_{B,F}, pk_{B,F}) \leftarrow$ KEYGEN(); $(sk_{B,R}, pk_{B,R}) \leftarrow$ KEYGEN()
else // real world
    $State^A \leftarrow$ WAITING FOR OPENING KEYS
    send (OPEN CHANNEL, 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 $State^B =$ 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
        $State^B \leftarrow$ WAITING FOR COMM SIG
    else // opening virtual channel
        $State^B \leftarrow$ WAITING FOR CHECK KEYS
    end if
    reply (ACCEPT CHANNEL, $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$)
    ensure $State^A =$ WAITING FOR OPENING KEYS
    store $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$
    $State^A \leftarrow$ OPENING KEYS OK
end if

Fig. 8.

Process $\textsc{ln.prepareBase}()$

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

Fig. 9.

Process LN.EXCHANGEOPENSIGS()

$C_{A,0} \leftarrow \mathrm{TX}\ \{\text{input: } (c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\}), \text{ outputs: } (c, (pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}), (0, pk_{B,\text{out}})\}$
$C_{B,0} \leftarrow \mathrm{TX}\ \{\text{input: } (c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\}), \text{ outputs: } (c, pk_{A,\text{out}}), (0, (pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})\}$
$\mathrm{sig}_{A,C,0} \leftarrow \textsc{sign}(C_{B,0}, sk_{A,F})$
$State^A \leftarrow$ WAITING FOR COMM SIG
send (FUNDING CREATED, $(c, pk_{A,\text{chain}})$, $\mathrm{sig}_{A,C,0}$) to `fundee`
ensure $State^B =$ WAITING FOR COMM SIG // if opening virtual channel, we have received (FUNDED, `host_fundee`) by `hops`[-1].`right` (c.f. Fig 12, line ??)
if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
  $F \leftarrow \mathrm{TX}\ \{\text{input: } (c, pk_{A,\text{chain}}), \text{ output: } (c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\})\}$
end if
$C_{B,0} \leftarrow \mathrm{TX}\ \{\text{input: } (c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\}), \text{ outputs: } (c, pk_{A,\text{out}}), (0, (pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})\}$
ensure $\textsc{verify}(C_{B,0}, \mathrm{sig}_{A,C,0}, pk_{A,F}) = \text{True}$
$C_{A,0} \leftarrow \mathrm{TX}\ \{\text{input: } (c, 3 \wedge 2/\{pk_{A,F}, pk_{B,F}\}), \text{ outputs: } (c, (pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}), (0, pk_{B,\text{out}})\}$
$\mathrm{sig}_{B,C,0} \leftarrow \textsc{sign}(C_{A,0}, sk_{B,F})$
if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
  $State^B \leftarrow$ WAITING TO CHECK FUNDING
else // opening virtual channel
  $c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
  $State^B \leftarrow$ OPEN
end if
reply (FUNDING SIGNED, $\mathrm{sig}_{B,C,0}$)
ensure $State^A =$ WAITING FOR COMM SIG
ensure $\textsc{verify}(C_{A,0}, \mathrm{sig}_{B,C,0}, pk_{B,F}) = \text{True}$

Fig. 10.

Process LN.COMMITBASE()

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

Fig. 11.

Process LN – external open messages for *Bob*

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

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

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

Fig. 12.

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

// `fundee` is *Bob*
ensure $P = Alice$ // activated party is the funder
if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then** // opening base channel
    ensure $State^A =$ TOPPED UP
    ensure $c = c_{A,\text{chain}}$
else // opening virtual channel
    ensure len(`hops`) ≥ 2 // cannot open a virtual over 1 channel
end if
LN.EXCHANGEOPENKEYS()
LN.PREPAREBASE()
if real world **then**
    LN.EXCHANGEOPENSIGS()
else // ideal world
    send (FUNDING CREATED, (c, $pk_{A,\text{chain}}$)) to $\mathcal{A}$ and expect reply (FUNDING CREATED OK)
end if
if `hops` $= \mathcal{G}_{\text{Ledger}}$ **then**
    LN.COMMITBASE()
else if ideal world **then** // opening virtual channel in ideal world
    $State^B \leftarrow$ OPEN
end if
$c_A \leftarrow c$; $c_B \leftarrow 0$; $i \leftarrow 0$
$State^A \leftarrow$ OPEN
output (OPEN OK, c, `fundee`, `hops`)

Fig. 13.

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
$\text{sig}_{P,C,i+1} \leftarrow$ 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
$\text{sig}_{\bar{P},C,i+1} \leftarrow$ 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. 14.

Process LN – virtualise start and end

On input (FUND ME, `fundee`, `hops`, c_{guest}, $pk_{A,V}$, $pk_{B,V}$) by `funder`:

  ensure $State^P$ = OPEN

  ensure $c_P - \texttt{locked}_P \geq c$

  $State^P \leftarrow$ 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()

  $State^P \leftarrow$ 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 $\texttt{host}'_P$: // real or ideal world

  ensure $State^P$ = WAITING FOR HOSTS READY

  $State^P \leftarrow$ OPEN

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

  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, $\texttt{host}_P$) 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

    output (FUNDED, $\texttt{host}_P$) 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. 15.

Process LN – virtualise hops

On (VIRTUALISING, $\texttt{host}'_{\bar{P}}$, $pk'_{\bar{P},F}$, $\texttt{hops}$, $\texttt{fundee}$, c_{guest}) by $\bar{P}$:
    ensure $State^P = \text{OPEN}$
    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
    $State^P \leftarrow$ 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
    $State^P \leftarrow$ 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}, $c_{\text{sib,rem}}$, sib) by $\texttt{sibling}$:
    ensure $State^P = \text{OPEN}$
    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
    $State^P \leftarrow$ VIRTUALISING
    $\texttt{locked}_P \leftarrow \texttt{locked}_P + c$
    define new VIRT ITI $\texttt{host}'_P$
    send (VIRTUALISING, $\texttt{host}'_P$, $pk'_{P,F}$, $\texttt{hops}$, $\texttt{fundee}$, 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)
    $State^P \leftarrow$ WAITING FOR REVOCATION
    output (VIRTUALISING ACK, $\texttt{host}'_P$, $pk'_{\bar{P},F}$) to $\texttt{sibling}$

Fig. 16.

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
if `host`$_{\bar{P}} \neq \mathcal{G}_{\text{Ledger}} \land \bar{P}$ 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
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
$R_{P,i} \leftarrow \text{TX}$ {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. 17.

Process LN.REVOCATIONSTRIP()

ensure VERIFY($C_{P,i+1}$, $\mathrm{sig}_{\bar{P},C,i+1}$, $pk_{\bar{P},F}$) = True
$R_{P,i} \leftarrow$ TX {input: $C_{\bar{P},i}$.outputs.P, output: ($c_{\bar{P}}$, $pk_{P,\mathrm{out}}$)}
ensure VERIFY($R_{P,i}$, $\mathrm{sig}_{\bar{P},R,i}$, $pk_{\bar{P},R}$) = True
$R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: (c_P, $pk_{\bar{P},\mathrm{out}}$)}
$\mathrm{sig}_{P,R,i} \leftarrow$ 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 `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 `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, $\mathrm{sig}_{P,R,i}$) to $\bar{P}$
$R_{\bar{P},i} \leftarrow$ TX {input: $C_{P,i}$.outputs.$\bar{P}$, output: (c_P, $pk_{\bar{P},\mathrm{out}}$)}
ensure VERIFY($R_{\bar{P},i}$, $\mathrm{sig}_{P,R,i}$, $pk_{P,R}$) = 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 `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 `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. 18.

Process LN – On (PAY, x):

ensure $State^{P}$ = OPEN $\wedge c_P \geq x$
if `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 stays idle forever
end if
if ideal world **then**
    send (PAY, x) to $\mathcal{A}$ and expect reply (PAY OK)
    ensure $State^{\bar{P}}$ = OPEN
    $c_P \leftarrow c_B - x; c_{\bar{P}} \leftarrow c_{\bar{P}} + x$; $i \leftarrow i + 1$
    if `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 `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. 19.

Process LN – On (CHECK FOR LATERAL CLOSE):

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

Fig. 20.

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

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

Fig. 21.

Process LN.SUBMITANDCHECKREVOCATION(j)

$\text{sig}_{P,R,j} \leftarrow \text{SIGN}(R_{P,j}, sk_{P,R})$
input (SUBMIT, $(R_{P,j}, \text{sig}_{P,R,j}, \text{sig}_{\bar{P},R,j})$) to $\mathcal{G}_{\text{Ledger}}$ as P
while $\nexists R_{P,j} \in \Sigma$ **do**
    wait for input (CHECK REVOCATION) // ignore other messages
    input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign output to Σ
end while

Fig. 22.

Process LN – On (CLOSE):

ensure $\mathit{State}^P \notin \{\bot, \text{INIT}, \text{TOPPED UP}\}$ // channel open
if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then** // we have a virtual channel
    $\mathit{State}^P \leftarrow$ 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 $\texttt{host}_P$
end if
$\mathit{State}^P \leftarrow$ CLOSING
input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P 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 $\mathcal{G}_{\text{Ledger}}$ as P and assign output to Σ
    end while
    // provably reachable – c.f. 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 $\mathcal{G}_{\text{Ledger}}$ as P
    while $C_{P,i} \notin \Sigma$ **do**
        wait for input (CHECK CLOSED) // ignore other messages
        input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign output to Σ
    end while
    // provably reachable – c.f. TODO: ref
end if
$\mathit{State}^P \leftarrow$ CLOSED
output (CLOSED)

Fig. 23.

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

Process VIRT

On input (INIT, `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
  output (HOST INIT OK) to `guest`

On input (HOST ME, `funder`, `fundee`, $\bar{P}$, `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`:
  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`
      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. 24.

Process VIRT

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\quad \text{TX}_{\text{both}} \leftarrow \text{TX}$ {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: $((c_{\text{left}} - c_{\text{guest}}, \texttt{3} \land \texttt{left\_virt}), (c_{\text{right}} - c_{\text{guest}}, \texttt{3} \land \texttt{right\_virt}), (c_{\text{guest}}, pk_{\text{loc,out}})$}

$\quad$ **return** $(\text{TX}_{\text{none}}, \text{TX}_{\text{left}}, \text{TX}_{\text{right}}, \text{TX}_{\text{both}})$

Fig. 25.

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}, $pk_{\text{left,fund}}$, $pk_{\text{right,fund}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, $\{pk_{\text{sec},i}\}_{i \in 1 \dots n}$, `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` $\land$ `virt_checked`) $\lor$ (`3` $\land$ `virt` $+ t$)

  else // TXs belong to `fundee`

    `out` $\leftarrow$ (`2` $\land$ `virt_checked`) $\lor$ (`3` $\land$ `virt` $+ t$)

  end if

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

  `guest_out` $\leftarrow$ `4` $\land\ 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

  `guest` $\leftarrow$ (`guest_out` $+ t$) $\lor$ `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. 26.

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($c_{\text{guest}}, c_P, c_{\bar{P}}, c_{\text{sib}}, c_{\text{sib,rem}}, pk_{\text{sib,rem},F}, pk_{\text{sib},F}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{sib,rem,virt}}, pk_{\text{loc,virt}}, pk_{\text{loc,virt}}, pk_{\text{rem,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}, pk_{\text{loc,virt}}, \{pk_{\text{sec},i}\}_{i \in 1 \ldots n}$)
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($c_{\text{guest}}, c_P, c_{\bar{P}}, pk_{\text{loc},F}, pk_{\text{rem},F}, pk_{\text{loc,virt}}, pk_{\text{rem,virt}}, pk_{\text{left,guest}}, pk_{\text{right,guest}}$, 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. 27.

Process VIRT.INTERMEDIARYSIGS()

$(\mathrm{TX}_{\mathrm{loc,none}}, \mathrm{TX}_{\mathrm{loc,left}}, \mathrm{TX}_{\mathrm{loc,right}}, \mathrm{TX}_{\mathrm{loc,both}}) \leftarrow$ VIRT.GETMIDTXS(c_{guest}, c_P, $c_{\bar{P}}$, c_{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 (VIRTUAL SIGS FORWARD, $\mathrm{sigs}_{\mathrm{byLeft}}$) to `sibling`

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`

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

end if

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

Fig. 28.

Process VIRT.HOSTFUNDEESIGS()

$\mathrm{TX}_{\mathrm{loc,none}} \leftarrow$ VIRT.GETEDGETXS(c_{guest}, c_P, $c_{\bar{P}}$, $pk_{\mathrm{loc},F}$, $pk_{\mathrm{rem},F}$, $pk_{\mathrm{loc,virt}}$, $pk_{\mathrm{rem,virt}}$, $pk_{\mathrm{left,guest}}$, $pk_{\mathrm{right,guest}}$, 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 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 $pk_{i,\mathrm{virt}}$

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

  add SIGN($\mathrm{TX}_{i,\mathrm{right}}$, $sk_{\mathrm{loc,virt}}$, ANYPREVOUT) and SIGN($\mathrm{TX}_{i,\mathrm{both}}$, $sk_{\mathrm{loc,virt}}$, ANYPREVOUT) to $\mathrm{sigs}_{\mathrm{toLeft}}$ // 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

  output (SIGN TXS, $\mathrm{TX}_{i,\mathrm{left}}$, $\mathrm{TX}_{i,\mathrm{right}}$, $\mathrm{TX}_{i,\mathrm{both}}$) to `fundee` and expect reply (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 SIGN($\mathrm{TX}_{i,\mathrm{left}}$, $sk_{\mathrm{loc,virt}}$, ANYPREVOUT) to $\mathrm{sigs}_{\mathrm{toLeft}}$

  end if

end for

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

Fig. 29.

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 $\{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n}$ // $\{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n}$ contains each party's $pk_{i,\mathrm{virt}}$
  ($\mathrm{TX}_{i,\mathrm{none}}$, $\mathrm{TX}_{i,\mathrm{left}}$, $\mathrm{TX}_{i,\mathrm{right}}$, $\mathrm{TX}_{i,\mathrm{both}}$) $\leftarrow$ VIRT.GETMIDTXS($\texttt{data}_i$, $\{pk_{\mathrm{sec},i}\}_{i \in 1 \dots n}$)
  add SIGN($\mathrm{TX}_{i,\mathrm{right}}$, $sk_{\mathrm{loc},\mathrm{virt}}$, ANYPREVOUT) and SIGN($\mathrm{TX}_{i,\mathrm{both}}$, $sk_{\mathrm{loc},\mathrm{virt}}$, ANYPREVOUT) to $\mathrm{sigs}_{\mathrm{toRight}}$ // 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, $\mathrm{TX}_{i,\mathrm{left}}$, $\mathrm{TX}_{i,\mathrm{right}}$, $\mathrm{TX}_{i,\mathrm{both}}$) to `fundee` and expect reply (TXS SIGNED, $\mathrm{sigs}_{\mathrm{guest}}$)
  add $\mathrm{sigs}_{\mathrm{guest}}$ to $\mathrm{sigs}_{\mathrm{toRight}}$
  if $i = 2$ **then** // hop is our next
    add SIGN($\mathrm{TX}_{i,\mathrm{left}}$, $sk_{\mathrm{loc},\mathrm{virt}}$, ANYPREVOUT) to $\mathrm{sigs}_{\mathrm{toRight}}$
  end if
end for
call VIRT.CIRCULATEVIRTUALSIGS($\mathrm{sigs}_{\mathrm{toRight}}$) of P and assign output to $\mathrm{sigs}_{\mathrm{byRight}}$
$\mathrm{TX}_{\mathrm{loc},\mathrm{none}} \leftarrow$ VIRT.GETEDGETXS(c_{guest}, c_P, $c_{\bar{P}}$, $pk_{\mathrm{loc},F}$, $pk_{\mathrm{rem},F}$, $pk_{\mathrm{loc},\mathrm{virt}}$, $pk_{\mathrm{rem},\mathrm{virt}}$, $pk_{\mathrm{left},\mathrm{guest}}$, $pk_{\mathrm{right},\mathrm{guest}}$, True)
return (OK)

Fig. 30.

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

if $\mathrm{sigs}_{\mathrm{byLeft}}$ 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. 31.

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

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

Fig. 32.

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()
    call virt.circulateRevocations(revoc_for_next) of P̄ and assign returned value to revoc_by_next
    ensure guest.processRemoteRevocation(revoc_by_next) returns (ok)
    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()
    call virt.circulateRevocations(revoc_for_next) of P̄ 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. 33.

Process VIRT – poll

On input (CHECK FOR LATERAL CLOSE) by $R \in \{$`guest`, `funder`, `fundee`$\}$:
  input (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign output to Σ
  `prev_went_on_chain` $\leftarrow \text{TX}_{\text{prev,left}} \in \Sigma \wedge \text{TX}_{\text{prev,both}} \in \Sigma$
  `next_went_on_chain` $\leftarrow \text{TX}_{\text{next,right}} \in \Sigma \wedge \text{TX}_{\text{next,both}} \in \Sigma$
  if `prev_went_on_chain` $\vee$ `next_went_on_chain` **then**
    ignore all messages except for (CHECK CHAIN FOR CLOSING) by R
    $State \leftarrow$ CLOSING
  end if
  if `prev_went_on_chain` $\wedge$ `next_went_on_chain` **then**
    VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,both}}$, $\text{sigs}_{\text{loc,both}}$)
  else if `prev_went_on_chain` **then**
    VIRT.SIGNANDSUBMIT($\text{TX}_{\text{loc,left}}$, $\text{sigs}_{\text{loc,left}}$)
  else if `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. 34.

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
    end if
    ensure *State* = OPEN
    if $\texttt{host}_P \neq \mathcal{G}_{\text{Ledger}}$ **then** // `host` is a VIRT
        ignore all messages except for output (CLOSED) by `host`. 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

Fig. 35.

1 Security Proof

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

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

Moving on, in the ideal world subsequently lines **??**-**??** of Fig. **??** are executed, which results in $\mathcal{S}$ executing lines 22-25 of Fig. 4. During the latter steps, $\mathcal{S}$ simulates executing line **??** of Fig. **??** with *Alice*.

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

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

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

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

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

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

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

$$
\begin{aligned}
\text{logged-coins}(P) &= c_P, \textit{ as recorded in } \mathcal{F}_{\text{Chan}}/\Pi_{\text{Chan}} \\
\text{ledger-coins}(P) &= \textit{coins spendable with the secret key sk of } P \textit{ if the closing} \\
&\quad \textit{transactions of all open channels are submitted to } \mathcal{G}_{\text{Ledger}} \\
&\quad \textit{and added to the state of all parties and then } t \textit{ new blocks} \\
&\quad \textit{enter the state of all honest parties} \\
\text{top-up}(P) &= \begin{cases} c_{\text{on}}, \textit{ as determined on message } (\textsc{check top up}), \\ \qquad \textit{if such a message was handled} \\ 0, \qquad \textit{otherwise} \end{cases} \\
\text{pay-in}(m, P) &= \begin{cases} x, \qquad \textit{if message } m \textit{ updated the channel to} \\ \quad \textit{a state in which } P \textit{ had } x \textit{ more coins} \textit{TODO: improve prev} \\ 0, \qquad \textit{otherwise} \end{cases} \\
\text{pay-out}(m, P) &= \begin{cases} x, \textit{ if } m = (\textsc{pay}, x) \textit{ was received by } P \textit{ and} \\ \quad P \textit{ output } (\textsc{pay success}) \textit{ as a result} \\ 0, \qquad \textit{otherwise} \end{cases}
\end{aligned}
$$

References