Both Π_{Chan} and $\mathcal{F}_{\text{Chan}}$ are parametrized by the stateful processes PCN (payment channel network) and VIRT (virtual layer). TODO: if the 2 processes share too much state, merge into 1 process

If:

- Π_{Chan} ($\mathcal{F}_{\text{Chan}}$) is activated by $\mathcal{E}$ ($Dave \in \{Alice, Bob\}$),
- Π_{Chan} ($\mathcal{F}_{\text{Chan}}$) then calls a method of either process (expecting some value to be returned by it),
- and subsequently the method gives up the execution token to another ITI (before it returns),

then Π_{Chan} ($\mathcal{F}_{\text{Chan}}$) repeatedly relays any input by $\mathcal{E}$ ($Dave$) to the method until the latter returns.

The following functions are available for both Π_{Chan} and $\mathcal{F}_{\text{Chan}}$.

locked coins in nested virtual channels

locked($Alice$):
  res $\leftarrow 0$
  for $((c_1, c_2), \_, \_, (Alice, \_), \_) \in$ `virtuals` **do**
    res $\leftarrow$ res $+ c_1 + c_2$
  end for
  return res

Fig. 1.

Protocol Π_{Chan}

On (INIT, `out_keys`) by $\mathcal{E}$:
    ensure $\mathit{State} = \bot$
    $(c_A, c_B) \leftarrow (0, 0)$
    `virtuals` $\leftarrow \emptyset$
    ensure PCN.INIT(`keys`, *Alice*) returns (OK)
    $\mathit{State} \leftarrow$ INIT

On TOP UP by $\mathcal{E}$, act like $\mathcal{F}_{\text{Chan}}$ (Fig. 4, lines 11-15)

On (OPEN BASE) by $\mathcal{E}$: TODO: LN.OPENBASE: `keys` $= pk_{A,out}, pk_{B,out}$
    ensure $\mathit{State} =$ TOPPED UP
    ensure PCN.OPENBASE(`keys`, `fundee`) returns (OK, c)
    $c_A \leftarrow c$; $c_B \leftarrow 0$
    $\mathit{State} \leftarrow$ OPEN BASE
    output (OPEN BASE SUCCESS) to $\mathcal{E}$

On (PAY, x) by $\mathcal{E}$:
    ensure $\mathit{State} \in \{$OPEN BASE, OPEN VIRTUAL$\}$
    ensure $c_A - \text{locked}(A) \geq x$
    ensure PCN.PAY(x) returns (OK)
    $c_A \leftarrow c_A - x$; $c_B \leftarrow c_B + x$
    output (PAY SUCCESS) to $\mathcal{E}$

On (BALANCE) by $\mathcal{E}$, act like $\mathcal{F}_{\text{Chan}}$ (Fig. 5, lines 15-16)

On (CLOSE) by $\mathcal{E}$, act like $\mathcal{F}_{\text{Chan}}$ (Fig. 5, lines 18-27):

Fig. 2.

Protocol Π_{Chan} – virtual

// notification to fundee
// trust that *Charlie* has c in her channel
On input (OPEN VIRTUAL, c, *Bob*, `host_bob`) by *Charlie*:
  ensure *State* = INIT
  ensure PCN.OPENVIRTUAL(*Bob*, *Charlie*, `host_bob`, c) returns (OK, `keys`)
  `host_alice` ← *Charlie*
  $c_A \leftarrow c$; $c_B \leftarrow 0$
  from now on, handle any (RELAYED, m) input by `host_alice` as the input (m) by $\mathcal{E}$
  from now on, transform any output (m) to $\mathcal{E}$ to output (RELAY, m) to `host_alice`
  State ← OPEN VIRTUAL
  output (OK, `keys`) to *Charlie*

On (FUND, c, hops, `inner_parties` = (`funder`, `fundee`), `outer_parties` = (`host_funder`, `host_fundee`)) by $\mathcal{E}$:
  ensure *State* $\in$ {OPEN BASE, OPEN VIRTUAL}
  ensure $c_A - \text{locked}(A) \geq c$
  do the same as in $\mathcal{F}_{\text{Chan}}$, Fig. 6, lines 2-8, skipping line 4 and replacing "to *Alice*" with "to $\mathcal{E}$" // "as *Alice*" sender labels are applied anyway, since we *are* *Alice*

On (RELAY, m, *Charlie*) by $\mathcal{E}$:
  do the same as in $\mathcal{F}_{\text{Chan}}$, Fig. 6, lines 23-24

On output (RELAY, m) by *Charlie*:
  do the same as in $\mathcal{F}_{\text{Chan}}$, Fig. 6, lines 26-27
TODO: check that everything done in ideal wrt closing is also done here

Fig. 3.

TODO: Add support for cooperative adding multiple virtuals to single channel (needs cooperation by all hops of all existing virtuals of current channel) TODO: Add support for cooperative closing (for virtual it also needs cooperation with all hops of all existing virtuals, we should definitely find another way)

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

On (INIT, `out_keys`) by *Alice*:
- ensure $State \in \{\perp, \text{INIT}_{Bob}\}$
- $(c_A, c_B) \leftarrow (0, 0)$
- `virtuals` $\leftarrow \emptyset$
- ensure PCN.INIT(`keys`, *Alice*) returns (OK)
- **if** $State = \perp$ **then**
  - $State \leftarrow \text{INIT}_{Bob}$
- **else** // $State = \text{INIT}_{Bob}$
  - $State \leftarrow$ INIT
- **end if**

On (TOP UP) by *Alice*:
- ensure $State =$ INIT
- ensure PCN.TOPUP(*Alice*) returns (OK, c_{chain})
- $State \leftarrow$ TOPPED UP
- output (TOP UP SUCCESS) to *Alice*

On (CORRUPT) by P, addressed to *Alice*:
- ensure $P \in \{$`host_alice`$, \mathcal{A}\}$
- `virtual_secrets` $\leftarrow \emptyset$
- **for all** $(\_, \_, (\text{fundee}, \_), (Alice, \_), vid) \in$ `virtuals` **do**
  - send (CORRUPT) to fundee and ensure reply is (CORRUPTED, `secrets`)
  - append (`secrets`, *vid*) to `virtual_secrets`
- **end for**
- from now on, allow $\mathcal{A}$ to handle all *Alice*'s messages, i.e. act as a relay
- **if** *Bob* is not corrupted **then**
  - from now on, handle all messages by *Bob* as Π_{Chan} (Fig. 2-3)
- **end if**
- **if** $P =$ `host_alice` **then**
  - output (CORRUPTED, (LN.SECRETS(*Alice*), `virtual_secrets`)) to `host_alice`
- **else** // $P = \mathcal{A}$
  - send (CORRUPTED, (LN.SECRETS(*Alice*), `virtual_secrets`)) to $\mathcal{A}$
- **end if**

Fig. 4.

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

On (OPEN BASE, `fundee`) by *Alice*:
    ensure *State* = TOPPED UP
    Bob ← `fundee`
    ensure PCN.OPENBASE(*Bob*) returns (OK, c)
    $c_A \leftarrow c$; $c_B \leftarrow 0$
    State ← OPEN BASE
    output (OPEN BASE SUCCESS) to *Alice*

On (PAY, x) by *Dave* $\in \{Alice, Bob\}$:
    ensure *State* $\in$ {OPEN BASE, OPEN VIRTUAL}
    ensure $c_D - \text{locked}(D) \geq x$
    send (PAY, x, *Dave*) to $\mathcal{A}$ and expect reply (OK) TODO: decide if PCN.PAY() needed – probably not TODO: there is a problem with who returns – last message goes to payee, so control is not on our side and adding the last message would add 1 more purely technical attack vector and an unneeded round
    $c_D \leftarrow c_D - x; c_{\bar{D}} \leftarrow c_{\bar{D}} + x$ // $\bar{D}$ is *Alice* if D is *Bob* and vice-versa
    output (PAY SUCCESS) to *Dave*

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

On (CLOSE) by *Alice*:
    ensure *State* $\in$ {OPEN BASE, OPEN VIRTUAL}
    ensure VIRT.CLOSE(*Alice*) returns (OK, $(\text{tx}_i, (\sigma_{ij})_j)_i$) // VIRT doesn't need to know if we are base or virtual
    if *State* = OPEN BASE **then**
        ensure PCN.CLOSE(*Alice*, $(\text{tx}_i, (\sigma_{ij})_j)_i$) returns (OK)
        State ← CLOSED
        output (CLOSE SUCCESS) to *Alice*
    else // *State* = OPEN VIRTUAL
        State ← CLOSED
        output (CLOSED VIRTUAL, $(\text{tx}_i, (\sigma_{ij})_j)_i$) to `host_alice` as *Alice*
    end if

On ((PEER) CLOSED VIRTUAL, $(\text{tx}_i, (\sigma_{ij})_j)_i$) by *Charlie*:
    ensure *State* $\in$ {OPEN BASE, OPEN VIRTUAL}
    ensure ((c_L, c_R), hops, (*Charlie*, *Dave*), (*Frank*, *George*), `keys`, *vid*) $\in$ `virtuals`, with *Frank* $\in \{Alice, Bob\}$ // no stored commitment TX in entry yet TODO: keys = $pk_{A,V}, pk_{B,V}$
    ensure VIRT.CLOSED(c_L, c_R, $(\text{tx}_i, (\sigma_{ij})_j)_i$) returns (OK)
    add message contents to `virtuals` entry
    TODO: decide if the following is needed: output ((PEER) CLOSED VIRTUAL, c_{left}, *vid*) to *George* if peer closed, else to *Frank* TODO: if the previous is needed, we need to calculate c_{left} in VIRT.CLOSED() and return it here

Fig. 5.

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

On (FUND, c, `hops`, `inner_parties` = (`funder`, `fundee`), `outer_parties` = (`host_funder`, `host_fundee`)) by *Alice*: // we fund another channel
  ensure *State* $\in$ {OPEN BASE, OPEN VIRTUAL}
  ensure $c_A - \text{locked}(A) \geq c$
  ensure `host_funder` = *Alice*
  generate unique *vid*
  ensure VIRT.FUND(c, `hops`, `inner_parties`, `outer_parties`, PCN, *vid*) returns (OK)
  add ((c, 0), `hops`, `inner_parties`, `outer_parties`, *vid*) to `virtuals`
  output (FUND SUCCESS) to *Alice*

On input (OPEN VIRTUAL, c, `fundee`, `host_fundee`) by `host_funder` to *Alice*: // *Alice* is funded by `host_funder`
  ensure *State* = INIT
  send (IS PARENT, `host_funder`, *Alice*) to $\mathcal{G}_{\text{Trust}}$ and ensure reply is (IS PARENT, `host_funder`, *Alice*, `true`) // ensure caller is trusted TODO: rethink if this is needless: we could just assume trust to the 1st person that asks us to OPEN VIRTUAL
  input (YOU ARE HOST, c, `funder`, `fundee`, `host_funder`) to `host_fundee`
  ensure $c_A \geq c$
  ensure we are not already supporting a virtual channel
  output (OK)
  ensure PCN.OPENVIRTUAL(`fundee`, `host_funder`, `host_fundee`, c) returns (OK, `keys`)
  $c_A \leftarrow c; c_B \leftarrow 0$
  from now on, handle any (RELAYED, m) input by {`host_funder`, `host_fundee`} as if it were input (m) by {*Alice*, *Bob*} respectively
  from now on, transform any output (m) to {*Alice*, *Bob*} to output (RELAY, m) to {`host_funder`, `host_fundee`} respectively
  State $\leftarrow$ OPEN VIRTUAL
  output (OK, `keys`) to `host_funder`

On (RELAY, m, *Charlie*) by *Alice*:
  ensure there is an entry in `virtuals` with *Alice* as host of funder and *Charlie* as fundee sub-party
  input (RELAYED, m) to *Charlie*

On output (RELAY, m) by *Charlie* to *Alice*:
  ensure there is an entry in `virtuals` with *Alice* as host of funder and *Charlie* as fundee sub-party // defensive check, may be redundant due to being subroutine respecting
  output (RELAYED, m, *Charlie*) to $\mathcal{E}$

Fig. 6.

Process LN – init

INIT(keys, *Dave*):
  ensure *Dave* = *Alice*
  $pk_{A,\mathrm{out}} \leftarrow$ keys
  return (OK)

TOPUP(funder): TODO: move to COMMON if more stuff fits there
  ensure super.*State* = INIT
  $(sk_{\mathrm{chain}}, pk_{\mathrm{chain}}) \leftarrow$ KEYGEN()
  output (PUBLIC KEY, pk_{chain}) to funder
  while $\nexists \mathrm{tx} \in \Sigma, c_{\mathrm{chain}} : (c_{\mathrm{chain}}, pk_{\mathrm{chain}}) \in \mathrm{tx.outputs}$ **do**
    wait[a] for input (CHECK TOP UP) by funder
    input (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ as funder and assign ouput to Σ
  end while
  base_output $\leftarrow (c_{\mathrm{chain}}, pk_{\mathrm{chain}})$
  return (OK, c_{chain})

// pure function except for hardcoded t, used by VIRT
GETCOMMTX(c_A, c_B, $pk_{A,F}$, $pk_{A,\mathrm{out}}$, $pk_{A,R}$, $pk_{B,F}$, $pk_{B,\mathrm{out}}$, $pk_{B,R}$):
  return TX {input: ($c_A + c_B$, $2/\{pk_{A,F}, pk_{B,F}\}$), outputs: ((c_A, $(pk_{A,\mathrm{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}$), ($c_B$, $pk_{B,\mathrm{out}}$))}

// used by VIRT
GETCOMMKEYS():
  return $(pk_{A,F}, pk_{A,\mathrm{out}}, pk_{A,R}), (pk_{B,F}, pk_{B,\mathrm{out}}, pk_{B,R})$

[a] while waiting, all other messages by *Dave* are ignored

Fig. 7.

Process LN − base

$\textsc{openBase}(\texttt{fundee})$:

ensure super.$State$ = TOPPED UP

$(sk_{A,F}, pk_{A,F}) \leftarrow \textsc{keyGen}()$; $(sk_{A,R}, pk_{A,R}) \leftarrow \textsc{keyGen}()$

if ideal world **then**

$(sk_{B,F}, pk_{B,F}) \leftarrow \textsc{keyGen}()$; $(sk_{B,R}, pk_{B,R}) \leftarrow \textsc{keyGen}()$

else // real world

send (OPEN BASE CHANNEL, c_{chain}, $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$) to `fundee`

// colored code is run by `fundee`. Validation is implicit

ensure super.$State$ = INIT // "super": storage of enclosing protocol

store $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$

$(sk_{B,F}, pk_{B,F}) \leftarrow \textsc{keyGen}()$; $(sk_{B,R}, pk_{B,R}) \leftarrow \textsc{keyGen}()$

reply (ACCEPT BASE CHANNEL, $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$)

store $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$

end if

$F \leftarrow$ TX {input: `base_output`, output: $(c_{\text{chain}}, 2/\{pk_{A,F}, pk_{B,F}\})$}

if real world **then**

$C_{A,0} \leftarrow$ TX {input: F.output, outputs: (c_{chain}, $(pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}$), (0, $pk_{B,\text{out}}$)}

$C_{B,0} \leftarrow$ TX {input: F.output, outputs: (c_{chain}, $pk_{A,\text{out}}$), (0, $(pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}$)}

$\text{sig}_{A,C,0} \leftarrow \textsc{sign}(C_{B,0}, sk_{A,F})$

send (FUNDING CREATED, `base_output`, $\text{sig}_{A,C,0}$) to `fundee`

// implicitly verify that this is a continuation of the previous exchange

$F \leftarrow$ TX {input: `base_output`, output: $(c_{\text{chain}}, 2/\{pk_{A,F}, pk_{B,F}\})$}

$C_{B,0} \leftarrow$ TX {input: F.output, outputs: (c_{chain}, $pk_{A,\text{out}}$), (0, $(pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}$)}

ensure $\textsc{verify}(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F})$ = True

$C_{A,0} \leftarrow$ TX {input: F.output, outputs: (c_{chain}, $(pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}$), (0, $pk_{B,\text{out}}$)}

$\text{sig}_{B,C,0} \leftarrow \textsc{sign}(C_{A,0}, sk_{B,F})$

reply (FUNDING SIGNED, $\text{sig}_{B,C,0}$)

ensure $\textsc{verify}(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F})$ = True

end if

$\text{sig}_F \leftarrow \textsc{sign}(F, sk_{\text{chain}})$

send (OPEN, c_{chain}, $pk_{A,\text{out}}$, $pk_{B,\text{out}}$, F, sig_F, `funder`) to $\mathcal{A}$

while $F \notin \Sigma$ **do**

wait for input (CHECK FUNDING) by `funder`

input (READ) to $\mathcal{G}_{\text{Ledger}}$ as `funder` and assign output to Σ

end while

return (OK, c_{chain})

Fig. 8.

Process LN – open virtual

OPENVIRTUAL(`fundee`, `host_funder`, `host_fundee`, c):

  if real world **then**

    do funding ceremony as in base channel (Fig. 8, lines 15-30) TODO: abstract better

    return $(pk_{A,F}, pk_{B,F})$

  end if

  return (OK)

Fig. 9.

Process LN – pay

PAY(x, *payid*): // *Alice* pays, *Bob* gets paid

  $C_{B,i+1} \leftarrow C_{B,i}$ with x coins moved from *Alice*'s to *Bob*'s output

  $\mathrm{sig}_{A,C,i+1} \leftarrow \mathrm{SIGN}(C_{B,i+1}, sk_{A,F})$ // kept by *Alice*

  send (PAY, x, $\mathrm{sig}_{A,C,i+1}$, *payid*) to *Bob*

  $C_{B,i+1} \leftarrow C_{B,i}$ with x coins moved from *Alice*'s to *Bob*'s output

  ensure $\mathrm{VERIFY}(C_{B,i+1}, \mathrm{sig}_{A,C,i+1}, pk_{A,F})$ = True

  $C_{A,i+1} \leftarrow C_{A,i}$ with x coins moved from *Alice*'s to *Bob*'s output

  $\mathrm{sig}_{B,C,i+1} \leftarrow \mathrm{SIGN}(C_{A,i+1}, sk_{B,F})$ // kept by *Bob*

  $R_{A,i+1} \leftarrow$ TX {input: $C_{B,i+1}$.outputs.*Alice*, output: $(c_B, pk_{A,\mathrm{out}})$}

  $\mathrm{sig}_{B,R,i+1} \leftarrow \mathrm{SIGN}(R_{A,i+1}, sk_{B,R})$

  reply (COMMITMENT SIGNED, $\mathrm{sig}_{B,C,i+1}$, $\mathrm{sig}_{B,R,i+1}$)

  $C_{A,i+1} \leftarrow C_{A,i}$ with x coins moved from *Alice*'s to *Bob*'s output

  ensure $\mathrm{VERIFY}(C_{A,i+1}, \mathrm{sig}_{B,C,i+1}, pk_{B,F})$ = True

  $R_{A,i+1} \leftarrow$ TX {input: $C_{B,i+1}$.outputs.*Alice*, output: $(c_B, pk_{A,\mathrm{out}})$}

  ensure $\mathrm{VERIFY}(R_{A,i+1}, \mathrm{sig}_{B,R,i+1}, pk_{B,R})$ = True

  $R_{B,i+1} \leftarrow$ TX {input: $C_{A,i+1}$.outputs.*Bob*, output: $(c_A, pk_{B,\mathrm{out}})$}

  $\mathrm{sig}_{A,R,i+1} \leftarrow \mathrm{SIGN}(R_{B,i+1}, sk_{A,R})$

  add (x, *payid*) to `paid_out`

  send (REVOKE AND ACK, $\mathrm{sig}_{A,R,i+1}$) to *Bob*

  $R_{B,i+1} \leftarrow$ TX {input: $C_{A,i+1}$.outputs.*Bob*, output: $(c_A, pk_{B,\mathrm{out}})$}

  ensure $\mathrm{VERIFY}(R_{B,i+1}, \mathrm{sig}_{A,R,i+1}, pk_{A,R})$ = True

  add (x, *payid*) to `paid_in`

Fig. 10.

Process LN – close

CLOSE(P, $(\mathrm{tx}_i, (\sigma_{ij})_j)_i$):

  TODO: also cover case when we are virtual

  $\mathit{State} \leftarrow (\textsc{closing}, P, (\mathrm{tx}_i)_i)$

  input ($\textsc{submit}$, $(\mathrm{tx}_i, (\sigma_{ij})_j)_i$) to $\mathcal{G}_{\mathrm{Ledger}}$ as P

On activation when $\mathit{State} = (\textsc{closing}, P, (\mathrm{tx}_i)_i)$:

  input ($\textsc{read}$) to $\mathcal{G}_{\mathrm{Ledger}}$ as P and assign output to Σ

  ensure all transactions $(\mathrm{tx}_i)_i$ are contained in Σ

  $\mathit{State} \leftarrow \textsc{closed}$

  return ($\textsc{ok}$)

Fig. 11.

Process VIRT

```
FUND(c, hops, (funder, fundee), (host_funder, host_fundee), PCN, vid):
    ensure host_funder = Alice // we are hosting the funder
    ensure len(hops) ≥ 2 // no point in opening a virtual over 1 channel
    send (OPEN VIRTUAL, c, fundee, host_fundee) to funder, ensure reply is (OK, (pk_{A,V}, pk_{B,V}))
    ensure VIRT.CIRCULATEVIRTUALKEYS(hops, (pk_{A,V}, pk_{B,V})) returns (OK)
    ensure VIRT.CIRCULATEVIRTUALSIGS(c, hops) returns (OK)
    ensure VIRT.CIRCULATEFUNDINGSIGS(hops) returns (OK)
    ensure VIRT.CIRCULATEREVOCATIONS(hops) returns (OK)
    return (OK)

CIRCULATEVIRTUALKEYS(hops, (pk_{A,V}, pk_{B,V}), left_virt_keys):
    // hops is a list of pid pairs
    (sk_loc, pk_loc) ← KEYGEN()
    if left_virt_keys is given as argument then // we are not host_funder
        if len(hops[1:]) ≥ 1 then // we are not host_fundee
            input (VIRTUALKEYSFORWARD, hops[1:], (pk_{A,V}, pk_{B,V}), left_virt_keys, (sk_loc, pk_loc), (pk_{A,F}, pk_{B,F})) to hops[1].Alice // sibling
            store inputs as left_virt_keys, (sk_loc, pk_loc), left_fund_keys
            call VIRT.CIRCULATEVIRTUALKEYS(hops, (pk_{A,V}, pk_{B,V}), (pk_loc, left_virt_keys[0])) of hops[0].Bob and assign output to right_virt_keys
            output (VIRTUALKEYSBACK, right_virt_keys, (pk_{A,F}, pk_{B,F}))
            store outputs as right_virt_keys, right_fund_keys
            return (pk_loc, right_virt_keys[0])
        else // we are host_fundee
            TODO: there is a problem with the base-virtual channel semantics, as a base channel becomes virtual itself when it facilitates another virtual. When it's solved, we may have to reconsider the fundee below.
            call PCN.GETCOMMKEYS() of fundee and assign output to (pk'_{A,V}, _, _), (pk'_{B,V}, _, _)
            ensure pk_{A,V} = pk'_{A,V} ∧ pk_{B,V} = pk'_{B,V}
            return pk_loc
        end if
    else // we are host_funder
        call VIRT.CIRCULATEVIRTUALKEYS(hops, (pk_{A,V}, pk_{B,V}), pk_loc) of hops[0].Bob and assign output to (pk_right, pk_{right,2})
        return (OK)
    end if
```

Fig. 12.

Process VIRT

CIRCULATEVIRTUALSIGS(c, `hops`, $\text{sigs}_{\text{byLeft}}$):

if $\text{sigs}_{\text{byLeft}}$ is given as argument **then** // we are not `host_funder`

(`comm_keys_loc`, `comm_keys_rem`) $\leftarrow$ PCN.GETCOMMKEYS()

create all TXs that need $\text{sigs}_{\text{byLeft}}$ TODO:

verify $\text{sigs}_{\text{byLeft}}$ on TXs TODO:

if len(`hops`[1:]) ≥ 1 **then** // we are not `host_fundee`

input (VIRTUALSIGSFORWARD, `hops`[1:], $\text{sigs}_{\text{byLeft}}$) to `hops`[1].*Alice* // sibling needs $\text{sigs}_{\text{byLeft}}$ for closing

(`comm_keys_loc`, `comm_keys_rem`) $\leftarrow$ PCN.GETCOMMKEYS()

($\text{TX}_{\text{next,none}}$, $\text{TX}_{\text{next,left}}$, $\text{TX}_{\text{next,right}}$, $\text{TX}_{\text{next,both}}$, C) $\leftarrow$ VIRT.GETMIDTXS(c, c_A, c_B, $pk_{A,F}$, $pk_{B,F}$, `right_fund_keys`, pk_{loc}, `right_virt_keys`[0], `right_virt_keys`[0], `right_virt_keys`[1], $pk_{A,V}$, $pk_{B,V}$, `right_virt_keys`[0], `comm_keys_loc`, `comm_keys_rem`)

sign them for $\text{sigs}_{\text{toRight}}$ TODO:

call VIRT.CIRCULATEVIRTUALSIGS(c, `hops`, $\text{sigs}_{\text{toRight}}$) of `hops`[0].*Bob* and assign output to $\text{sigs}_{\text{byRight}}$

create all TXs that need $\text{sigs}_{\text{byRight}}$ TODO:

verify $\text{sigs}_{\text{byRight}}$ on TXs TODO:

output (VIRTUALSIGSBACK, $\text{sigs}_{\text{byRight}}$) // sibling needs $\text{sigs}_{\text{byRight}}$ for closing

end if

else // we are `host_funder`

(`comm_keys_loc`, `comm_keys_rem`) $\leftarrow$ PCN.GETCOMMKEYS()

($\text{TX}_{\text{next,none}}$, $\text{TX}_{\text{next,left}}$, $\text{TX}_{\text{next,right}}$, $\text{TX}_{\text{next,both}}$, C) $\leftarrow$ VIRT.GETMIDTXS(c, c_A, c_B, $pk_{A,F}$, $pk_{B,F}$, `right_fund_keys`, pk_{loc}, `right_virt_keys`[0], `right_virt_keys`[0], `right_virt_keys`[1], $pk_{A,V}$, $pk_{B,V}$, `right_virt_keys`[0], `comm_keys_loc`, `comm_keys_rem`)

sign them for $\text{sigs}_{\text{toRight}}$ TODO:

call VIRT.CIRCULATEVIRTUALSIGS(c, `hops`, $\text{sigs}_{\text{toRight}}$) of `hops`[0].*Bob* and assign output to $\text{sigs}_{\text{byRight}}$

create all TXs that need $\text{sigs}_{\text{byRight}}$ TODO:

verify $\text{sigs}_{\text{byRight}}$ on TXs TODO:

return (OK)

end if

if $\text{sigs}_{\text{byLeft}}$ is given as argument **then** // we are not `host_funder`

create all left's TXs that need our sigs and sign them for $\text{sigs}_{\text{toLeft}}$ TODO:

return $\text{sigs}_{\text{forLeft}}$

end if

Fig. 13.

Process VIRT

GETMIDTXS(c_{guest}, c_{loc}, c_{rem}, $pk_{\text{left,fund}}$, $pk_{\text{loc,fund}}$, $pk_{\text{sib,fund}}$, $pk_{\text{right,fund}}$, $pk_{\text{left,virt}}$, $pk_{\text{loc,virt}}$, $pk_{\text{sib,virt}}$, $pk_{\text{right,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, $pk_{\text{loc,out}}$, `comm_keys_loc`, `comm_keys_rem`):

- ensure $c_{\text{sibRem}} \geq c_{\text{guest}} \wedge c_{\text{loc}} \geq c_{\text{guest}}$
- $c_{\text{left}} \leftarrow c_{\text{sib}} + c_{\text{sibRem}}$; $c_{\text{right}} \leftarrow c_{\text{loc}} + c_{\text{rem}}$
- `left_fund` $\leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{loc,fund}}\}$
- `right_fund` $\leftarrow 2/\{pk_{\text{sib,fund}}, pk_{\text{right,fund}}\}$
- `left_virt` $\leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}\}$
- `right_virt` $\leftarrow 2/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}\}$
- `guest` $\leftarrow 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
- `left_out` $\leftarrow$ (`2` $\wedge$ `left_virt`) $\vee$ (`3` $\wedge$ `left_virt` $+ t$)
- `right_out` $\leftarrow$ (`1` $\wedge$ `right_virt`) $\vee$ (`3` $\wedge$ `right_virt` $+ t$)
- `guest_out` $\leftarrow$ `guest_virt` $\vee$ (`guest_virt` $+ t$)
- $\text{TX}_{\text{none}} \leftarrow$ TX {inputs: ((c_{left}, `left_fund`), (c_{right}, `right_fund`)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, `left_out`), ($c_{\text{right}} - c_{\text{guest}}$, `right_out`), ($c_{\text{guest}}$, $pk_{\text{loc,out}}$), (c_{guest}, `guest_out`))}
- $\text{TX}_{\text{left}} \leftarrow$ TX {inputs: (($c_{\text{left}} - c_{\text{guest}}$, `1` $\wedge$ `left_virt`), (c_{right}, `right_fund`)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, `3` $\wedge$ `left_virt`), ($c_{\text{right}} - c_{\text{guest}}$, `right_out`), ($c_{\text{guest}}$, $pk_{\text{loc,out}}$))}
- $\text{TX}_{\text{right}} \leftarrow$ TX {inputs: ((c_{left}, `left_fund`), ($c_{\text{right}} - c_{\text{guest}}$, `2` $\wedge$ `right_virt`), outputs: (($c_{\text{left}} - c_{\text{guest}}$, `left_out`), ($c_{\text{right}} - c_{\text{guest}}$, `3` $\wedge$ `right_virt`), (c_{guest}, $pk_{\text{loc,out}}$))}
- $\text{TX}_{\text{both}} \leftarrow$ TX {inputs: (($c_{\text{left}} - c_{\text{guest}}$, `1` $\wedge$ `left_virt`), ($c_{\text{right}} - c_{\text{guest}}$, `2` $\wedge$ `right_virt`), (c_{guest}, `guest`)), outputs: (($c_{\text{left}} - c_{\text{guest}}$, `3` $\wedge$ `left_virt`), ($c_{\text{right}} - c_{\text{guest}}$, `3` $\wedge$ `right_virt`), ($c_{\text{guest}}, pk_{\text{loc,out}}$))}
- $C \leftarrow$ PCN.GETCOMMTX($c_{\text{loc}} - c_{\text{guest}}, c_{\text{rem}}$, `comm_keys_loc`, `comm_keys_rem`)
- **return** (TX_{none}, TX_{left}, TX_{right}, TX_{both}, C)

Fig. 14.

Process VIRT

CIRCULATEFUNDINGSIGS(`hops`):
  return (OK)

CIRCULATEREVOCATIONS(`hops`):
  return (OK)

CLOSE(P): TODO: continue TODO: handle arbitrarily nested virtuals (now we only handle one level and it leads to nested ifs

if both channel parties are honest **then**
  if `funded` $\neq \emptyset$ **then**
    TODO: prepare virtual layer TX V and its signature – careful, may be unneded!
    $C \leftarrow$ TX {input: V.output, outputs: $(c_A, pk_{A,\text{out}} + t), (c_B, pk_{B,\text{out}})$}
    $\text{sig}_{B,C} \leftarrow \text{SIGN}(C, sk_{B,V})$
    $\text{sig}_{A,C} \leftarrow \text{SIGN}(C, sk_{A,V})$
  else
    $C \leftarrow$ TX {input: F.output, outputs: $(c_A, pk_{A,\text{out}} + t), (c_B, pk_{B,\text{out}})$}
    $\text{sig}_{B,C} \leftarrow \text{SIGN}(C, sk_{B,F})$
    $\text{sig}_{A,C} \leftarrow \text{SIGN}(C, sk_{A,F})$
  end if
end if // if *Bob* is corrupted, we already have C and $\text{sig}_{B,C}$
State $\leftarrow$ CLOSED
if *State* = OPEN BASE **then**
  input (SUBMIT, $(V, \text{sig}_{A,V}, \text{sig}_{B,V})$, $(C, \text{sig}_{A,C}, \text{sig}_{B,C})$) to $\mathcal{G}_{\text{Ledger}}$
else // *State* = OPEN VIRTUAL
  if *Alice* is the one that had received (OPEN VIRTUAL, ...) by `opener` **then**
    `initiator` $\leftarrow$ `opener`; `other` $\leftarrow$ `outer_peer`
  else // *Bob* had received (OPEN VIRTUAL ...)
    `initiator` $\leftarrow$ `outer_peer`; `other` $\leftarrow$ `opener`
  end if
  if both parties are honest **then**
    if `funded` $\neq \emptyset$ **then**
      TODO: prepare virtual layer TXs V_A, V_B and their signatures
      $C' \leftarrow$ TX {input: V.output, outputs: $(c_A, pk_{A,\text{out}}), (c_B, pk_{B,\text{out}} + t)$}
      $\text{sig}'_{A,C} \leftarrow \text{SIGN}(C', sk_{A,V})$; $\text{sig}'_{B,C} \leftarrow \text{SIGN}(C', sk_{B,V})$
    else // there are no virtual channels on top of us
      $C' \leftarrow$ TX {input: F.output, outputs: $(c_A, pk_{A,\text{out}}), (c_B, pk_{B,\text{out}} + t)$}
      $\text{sig}'_{A,C} \leftarrow \text{SIGN}(C', sk_{A,F})$; $\text{sig}'_{B,C} \leftarrow \text{SIGN}(C', sk_{B,F})$
    end if
    provide delayed output (PEER CLOSED VIRTUAL, $(V_B, \text{sig}_{A,V_B}, \text{sig}_{B,V_B})$, $(C', \text{sig}'_{A,C}, \text{sig}'_{B,C})$) to `other` as *Bob*
  end if
  if `funded` $\neq \emptyset$ **then**
    `output` $\leftarrow$ (CLOSED VIRTUAL, $(V, \text{sig}_{A,V}, \text{sig}_{B,V}), (C, \text{sig}_{A,C}, \text{sig}_{B,C})$)
  else
    `output` $\leftarrow$ (CLOSED VIRTUAL, $(C, \text{sig}_{A,C}, \text{sig}_{B,C})$)
  end if
  output `output` to `initiator` as *Alice*
end if

CLOSED(c_L, c_R, $(\text{tx}_i, (\sigma_{ij})_j)_i$):

  for all i in $1 \ldots |(\text{tx}_i, (\sigma_{ij})_j)_i|$ **do**
    ensure VERIFY$(\text{tx}_i, (\sigma_{ij})_j) = \text{True}$
  end for
  ensure $(\text{tx}_1, (\sigma_{1j})_j)$ has exactly 1 input, which spends an output of V of value $c_L + c_R$
  return (OK)

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

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

On ($\textsc{is parent}$, id_1, id_2) by P
  if id_1 is the parent of id_2 in G **then**
    send ($\textsc{is parent}$, id_1, id_2, true) to P
  else
    send ($\textsc{is parent}$, id_1, id_2, false) to P
  end if

Simulator $\mathcal{S}$– Pt. 1

```
On (OPEN, c_F, pk_{A,out}, pk_{B,out}, F, sig_F Alice) by F_Chan: // both honest
    simulate Alice receiving input (OPEN, c_F, pk_{A,out}, pk_{B,out}) by E
    ensure simulated Alice inputs (SUBMIT, (F', sig_{F'})) to G_Ledger
    input (SUBMIT, (F, sig_F)) to G_Ledger

On (OPEN, c_F, pk_{A,out}, pk_{B,out}, pk_{B,F}, Bob) by F_Chan: // Alice corrupted
    send LN message (OPEN, pk_{B,F}) to Alice and relay reply to F_Chan TODO: change msg to fit LN, ensure Alice doesn't see a difference from real world

On (PAY, x, Dave) by F_Chan:
    if both channel parties are honest then
        simulate Dave receiving input (PAY, x) by E
        ensure simulated Dave outputs (OK)
        send (OK) to F_Chan
    else if only Dave's counterparty is corrupted then // else just relay to A
        simulate Dave receiving input (PAY, x) by E
        ensure simulated Dave outputs (OK)
        extract the latest commitment transaction C and its signature by Dave's counterparty sig_{D̄,C} from simulated Dave's state
        send (C, sig_{D̄,C}) to F_Chan
    end if

On (FUND YOU, c, Bob, Charlie, Alice) by F_Chan:
    simulate Alice receiving input (FUND YOU, c, Bob) by Charlie
    ensure simulated Alice outputs (OK) to Charlie
    send (OK) to F_Chan

On (FUND c, hops, sub_parties = (fundee, counterparty), outer_parties = (Charlie, Dave), funder = Alice, id) by F_Chan:
    add the message data to virtual_opening
    simulate execution of line ?? of Fig. 6 with Alice// S knows Bob (Alice's counterparty) through opening procedure
    send (OK) to F_Chan

On (ALLOW FUND, c, sub_parties, local_funder = L_i, id, i =? |hops|) by F_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 F_Chan's Alice
```

Fig. 16.

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 15 of Fig. 3 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 15 of Fig. 3 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. 17.

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 1-**??** of Fig. 6 are executed and then control is handed over to the "fundee" ITI, which executes lines 3-11 of Fig. 3. This ITI will output (OK) if and only if line 5 of Fig. 3 succeeds.

When $\mathcal{E}$ sends (FUND, c, hops, (fundee, counterparty), (*Charlie*, *Dave*)) to *Alice* in the ideal world, lines 1-**??** of Fig. 6 are executed and then control is handed over to the functionality that controls the "fundee", which executes lines 9-**??** of Fig. 6 and then hands control over to $\mathcal{S}$. The latter in turn simulates lines 3-11 of Fig. 3, 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 5 of Fig. 3 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 5-**??** of Fig. 6 are executed, which results in $\mathcal{S}$ executing lines 22-25 of Fig. 16. During the latter steps, $\mathcal{S}$ simulates executing line **??** of Fig. 6 with *Alice*.

Similarly in the real world, *Alice* executes lines 5 and **??** of Fig. 6, therefore the two worlds still are perfectly indistinguishable.

The "for" loop of lines **??**-**??** of Fig. 6 is then executed in both the real and the ideal worlds. The message of line **??** results in the execution of lines 3-11 of Fig. 3 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. 6 prompts $\mathcal{S}$ to simulate line 15 of Fig. 3 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 **??**-7 of Fig. 6 is also perfectly indistinguishable in the two worlds. With argumentation similar to that of the previous "for" loop, we conclude that the FUND message does not induce any chance of distinguishability between the two worlds.

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

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

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

$$\begin{aligned}
\text{logged-coins}(P) &= c_P, \textit{ as recorded in } \mathcal{F}_{\text{Chan}}/\Pi_{\text{Chan}} \\
\text{ledger-coins}(P) &= \textit{coins spendable with the secret key sk of } P \textit{ if the closing} \\
&\quad \textit{transactions of all open channels are submitted to } \mathcal{G}_{\text{Ledger}} \\
&\quad \textit{and added to the state of all parties and then } t \textit{ new blocks} \\
&\quad \textit{enter the state of all honest parties}
\end{aligned}$$

$$\text{top-up}(P) = \begin{cases} c_{\text{on}}, \textit{ as determined on message } (\textsc{check top up}), \\ \qquad\qquad\qquad\qquad \textit{if such a message was handled} \\ 0, \qquad\qquad\qquad\qquad\qquad\qquad\qquad \textit{otherwise} \end{cases}$$

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

$$\text{pay-out}(m, P) = \begin{cases} x, \textit{ if } m = (\textsc{pay}, x) \textit{ was received by } P \textit{ and} \\ \quad P \textit{ output } (\textsc{pay success}) \textit{ as a result} \\ 0, \qquad\qquad\qquad\qquad\qquad\qquad \textit{otherwise} \end{cases}$$

References