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

Fig. 1.

Protocol Π_{Chan}

On (INIT, `out_keys`) by $\mathcal{E}$:
  ensure *State* $= \bot$
  $(c_A, c_B) \leftarrow (0, 0)$
  `virtuals` $\leftarrow \emptyset$
  ensure PCN.INIT(`keys`, *Alice*) returns (OK)
  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 *State* $=$ TOPPED UP
  ensure PCN.OPENBASE(`keys`, `fundee`) returns (OK, c)
  $c_A \leftarrow c$; $c_B \leftarrow 0$
  State $\leftarrow$ OPEN BASE
  output (OPEN BASE SUCCESS) to $\mathcal{E}$

On (PAY, x) by $\mathcal{E}$:
  ensure *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)
  `host_alice` $\leftarrow$ *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 $\leftarrow$ OPEN VIRTUAL
  output (OPEN VIRTUAL SUCCESS) 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 $c_A - \mathrm{locked}(A) \geq c$
  do the same as in $\mathcal{F}_{\mathrm{Chan}}$, 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 $\mathcal{F}_{\mathrm{Chan}}$, Fig. 6, lines 18-19

On output (RELAY, m) by *Charlie*:
  do the same as in $\mathcal{F}_{\mathrm{Chan}}$, Fig. 6, lines 21-22
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 $\mathcal{F}_{\text{Chan}}$ – 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 (_, _, (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
  ensure PCN.OPENVIRTUAL(`fundee`, `host_funder`, `host_fundee`, c) returns (OK)
  $c_A \leftarrow c; c_B \leftarrow 0$
  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) 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 $\textsc{ln}$ – init

$\textsc{init}(\texttt{keys}, \mathit{Dave})$:

  ensure $\mathit{Dave} = \mathit{Alice}$

  $pk_{A,\text{out}} \leftarrow \texttt{keys}$

  return $(\textsc{ok})$

$\textsc{topUp}(\texttt{funder})$: TODO: move to $\textsc{common}$ if more stuff fits there

  ensure super.$\mathit{State} = \textsc{init}$

  $(sk_{\text{chain}}, pk_{\text{chain}}) \leftarrow \textsc{keyGen}()$

  output $(\textsc{public key}, pk_{\text{chain}})$ to $\texttt{funder}$

  while $\nexists \text{tx} \in \Sigma, c_{\text{chain}} : (c_{\text{chain}}, pk_{\text{chain}}) \in \text{tx.outputs}$ **do**

    wait[a] for input $(\textsc{check top up})$ by $\texttt{funder}$

    input $(\textsc{read})$ to $\mathcal{G}_{\text{Ledger}}$ as $\texttt{funder}$ and assign ouput to Σ

  end while

  $\texttt{base\_output} \leftarrow (c_{\text{chain}}, pk_{\text{chain}})$

  return $(\textsc{ok}, c_{\text{chain}})$

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

Fig. 7.

Process LN − base

OPENBASE(`fundee`):
    ensure super.$State$ = TOPPED UP
    $(sk_{A,F}, pk_{A,F}) \leftarrow$ KEYGEN(); $(sk_{A,R}, pk_{A,R}) \leftarrow$ KEYGEN()
    if ideal world **then**
        $(sk_{B,F}, pk_{B,F}) \leftarrow$ KEYGEN(); $(sk_{B,R}, pk_{B,R}) \leftarrow$ 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 $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$
        $(sk_{B,F}, pk_{B,F}) \leftarrow$ KEYGEN(); $(sk_{B,R}, pk_{B,R}) \leftarrow$ 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}, ($pk_{A,\text{out}} \wedge$ `delay`) $\vee$ $(pk_{A,R} \wedge pk_{B,R})$), (0, $pk_{B,\text{out}}$)}
        $C_{B,0} \leftarrow$ TX {input: F.output, outputs: (c_{chain}, $pk_{A,\text{out}}$), (0, ($pk_{B,\text{out}} \wedge$ `delay`) $\vee$ $(pk_{A,R} \wedge pk_{B,R})$)}
        $\text{sig}_{A,C,0} \leftarrow$ 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, ($pk_{B,\text{out}} \wedge$ `delay`) $\vee$ $(pk_{A,R} \wedge pk_{B,R})$)}
        ensure VERIFY$(C_{B,0}, \text{sig}_{A,C,0}, pk_{A,F})$ = True
        $C_{A,0} \leftarrow$ TX {input: F.output, outputs: (c_{chain}, ($pk_{A,\text{out}} \wedge$ `delay`) $\vee$ $(pk_{A,R} \wedge pk_{B,R})$), (0, $pk_{B,\text{out}}$)}
        $\text{sig}_{B,C,0} \leftarrow$ SIGN$(C_{A,0}, sk_{B,F})$
        reply (FUNDING SIGNED, $\text{sig}_{B,C,0}$)
        ensure VERIFY$(C_{A,0}, \text{sig}_{B,C,0}, pk_{B,F})$ = True
    end if
    $\text{sig}_F \leftarrow$ SIGN(F, sk_{chain})
    send (OPEN, $c_{\text{chain}}, pk_{A,\text{out}}, pk_{B,\text{out}}, F, \text{sig}_F$, `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

```
VIRTUALKEYS(c, fundee, host_funder, host_fundee):
    ensure super.State = INIT // "super": storage of enclosing protocol
    store c, fundee, host_funder, host_fundee, vid
    (sk_{A,F}, pk_{A,F}) ← KEYGEN()
    if ideal world then
        (sk_{B,F}, pk_{B,F}) ← KEYGEN()
    else // real world
        send (REQUEST VIRTUAL KEY, pk_{A,F}, c, host_fundee) to fundee
        // colored code is run by fundee. Validation is implicit
        ensure super.State = INIT
        store pk_{A,F}, c, host_fundee
        (sk_{B,F}, pk_{B,F}) ← KEYGEN()
        reply (DELIVER VIRTUAL KEY, pk_{B,F})
    end if
    return (pk_{A,F}, pk_{B,F})

OPENVIRTUAL(funding_output):
    ensure funding_output = 2/{pk_{A,F}, pk_{B,F}}
    if real world then
        do funding ceremony as in base channel (Fig. 8, lines 15-30) TODO: abstract better
    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
  $\text{sig}_{A,C,i+1} \leftarrow \text{SIGN}(C_{B,i+1}, sk_{A,F})$ // kept by *Alice*
  send (PAY, x, $\text{sig}_{A,C,i+1}$, *payid*) to *Bob*
  $C_{B,i+1} \leftarrow C_{B,i}$ with x coins moved from *Alice*'s to *Bob*'s output
  ensure $\text{VERIFY}(C_{B,i+1}, \text{sig}_{A,C,i+1}, pk_{A,F}) = \text{True}$
  $C_{A,i+1} \leftarrow C_{A,i}$ with x coins moved from *Alice*'s to *Bob*'s output
  $\text{sig}_{B,C,i+1} \leftarrow \text{SIGN}(C_{A,i+1}, sk_{B,F})$ // kept by *Bob*
  $R_{A,i+1} \leftarrow \text{TX}$ {input: $C_{B,i+1}$.outputs.*Alice*, output: $(c_B, pk_{A,\text{out}})$}
  $\text{sig}_{B,R,i+1} \leftarrow \text{SIGN}(R_{A,i+1}, sk_{B,R})$
  reply (COMMITMENT SIGNED, $\text{sig}_{B,C,i+1}$, $\text{sig}_{B,R,i+1}$)
  $C_{A,i+1} \leftarrow C_{A,i}$ with x coins moved from *Alice*'s to *Bob*'s output
  ensure $\text{VERIFY}(C_{A,i+1}, \text{sig}_{B,C,i+1}, pk_{B,F}) = \text{True}$
  $R_{A,i+1} \leftarrow \text{TX}$ {input: $C_{B,i+1}$.outputs.*Alice*, output: $(c_B, pk_{A,\text{out}})$}
  ensure $\text{VERIFY}(R_{A,i+1}, \text{sig}_{B,R,i+1}, pk_{B,R}) = \text{True}$
  $R_{B,i+1} \leftarrow \text{TX}$ {input: $C_{A,i+1}$.outputs.*Bob*, output: $(c_A, pk_{B,\text{out}})$}
  $\text{sig}_{A,R,i+1} \leftarrow \text{SIGN}(R_{B,i+1}, sk_{A,R})$
  add (x, *payid*) to `paid_out`
  send (REVOKE AND ACK, $\text{sig}_{A,R,i+1}$) to *Bob*
  $R_{B,i+1} \leftarrow \text{TX}$ {input: $C_{A,i+1}$.outputs.*Bob*, output: $(c_A, pk_{B,\text{out}})$}
  ensure $\text{VERIFY}(R_{B,i+1}, \text{sig}_{A,R,i+1}, pk_{A,R}) = \text{True}$
  add (x, *payid*) to `paid_in`

Fig. 10.

Process LN – close

CLOSE(P, $(\text{tx}_i, (\sigma_{ij})_j)_i$):
  TODO: also cover case when we are virtual
  $State \leftarrow (\text{CLOSING}, P, (\text{tx}_i)_i)$
  input (SUBMIT, $(\text{tx}_i, (\sigma_{ij})_j)_i$) to $\mathcal{G}_{\text{Ledger}}$ as P

On activation when $State = (\text{CLOSING}, P, (\text{tx}_i)_i)$:
  input (READ) to $\mathcal{G}_{\text{Ledger}}$ as P and assign reply to Σ
  ensure all transactions $(\text{tx}_i)_i$ are contained in Σ
  $State \leftarrow \text{CLOSED}$
  return (OK)

Fig. 11.

Process VIRT

FUND(c, `hops`, (`funder`, `fundee`), (`host_funder`, `host_fundee`), PCN, vid):
    TODO: do VChan() with hops – $P_{i-1}P_i, P_iP_{i+1}$ and all P_1P_n held by BOTH R_{i-1} and L_i. $P_{i-1}P_i$ held only by R_{i-1}, P_iP_{i+1} held only by L_i. This (probably) ensures that only relevant parties can close their channels (with the exception of honest R_{i-1} wanting to leave channels virtual but corrupted L_i demoting them to base, which however doesn't cost funds to anyone), but that they have minimal impact to the decisions of ajdacent channels. All $P_{i-1}P_i$ inputs must be signed by R_{i-1} and all P_iP_{i+1} inputs by L_i.
ensure `host_funder` $=$ *Alice* // we are hosting the funder
$(pk_{A,V}, pk_{B,V}) \leftarrow$ PCN.VIRTUALKEYS(c, `fundee`, `host_funder`, `host_fundee`)
$C_{\text{temp}} \leftarrow C_i$ with c coins moved from *Alice*'s output to a new $2/\{pk_{A,V}, pk_{B,V}\}$ output named `virtual_output` TODO: (optional:) make more formal
ensure PCN.OPENVIRTUAL(`funding_output`) returns (OK)
TODO: continue: send relevant data, including signatures for next comm TX, only to the next hop. Enter a waiting state, expecting reply from next hop with all signatures needed for next comm TX and the revocation sig of the previous comm TX. Mark virtual channel as open. Send to next hop the revocation sig. While waiting, ignore everything else apart from channel closing
TODO: Think through the differences of the ideal world, e.g. that if some continuous hops are honest, we only need per-hop *confirmation* by $\mathcal{S}$ (which probably simulates an honest exchange normally), whereas for hops from honest to corrupt we do the whole drill with $\mathcal{S}$ (which probably does the corrupt part of the exchange according to $\mathcal{A}$'s instructions)
TODO: Add logic for intermediaries and fundee

$(L_0, R_0) \leftarrow (Alice, Bob)$
for all $(P, pk) \in$ `hops` **do** // $i \in \{1, \dots, |\texttt{hops}|\}$
    send (ALLOW FUND, c, `sub_parties`, vid, $i \stackrel{?}{=} |\texttt{hops}|$) to P as *Alice* and ensure reply is (OK)
end for
if both channel parties are honest **then**
    send (IS OPEN SUCCESSFUL, vid) to $\mathcal{A}$ and ensure reply is (OK)
else if only *Alice* is honest **then**
    $(sk_{A,V}, pk_{A,V}) \leftarrow$ KEYGEN()
    send (UPDATE TO VIRTUAL, $pk_{A,V}$) to $\mathcal{A}$ and assign reply to ($V =$ TX {input: F.output, outputs: $(c_A + c_B - c, 2/\{pk_{A,V}, pk_{B,V}\})$, $(c, 2/\{pk_{G,V}, pk_{A,V}\})$, $(0, |\texttt{hops}|/\{\texttt{hops}_i.pk\}_i)$}, $\text{sig}_{B,V}$, $C' =$ TX {input: V.outputs.0, outputs: $(c_A - \text{locked}_A - c, pk_{A,\text{out}} \wedge t), (c_B - \text{locked}_B, pk_{B,\text{out}})$}, $\text{sig}_{B,C'}$) TODO: think about locked coins
    ensure VERIFY(V, $\text{sig}_{B,V}$, $pk_{B,F}$) $=$ VERIFY(C', $\text{sig}_{B,C'}$, $pk_{A,V}$) $=$ True
end if
for all $(P, pk) \in$ `hops` **do** // $i \in \{1, \dots, |\texttt{hops}|\}$
    send (FUND DONE, vid) to P as *Alice* and ensure reply is (OK)
end for
$c_A \leftarrow c_A - c$
if only *Alice* is honest **then**
    $C \leftarrow C'$; $\text{sig}_{B,C} \leftarrow \text{sig}_{B,C'}$
end if
// notification to hop that locks coins
On (ALLOW FUND, c, `sub_parties`, `next_hop`, id, `is_last`) by *Charlie*:
    ensure *State* $\in$ {OPEN BASE, OPEN VIRTUAL}
    ensure $c_A - \text{locked}(A) \geq c$
    ensure *Bob* belongs to the same group as `next_hop`
    output received message to *Dave* and ensure reply is (OK)
    send (ALLOW FUND, c, `sub_parties`, `next_hop`, id, `is_last`, *Charlie*) to *Bob* and ensure reply is (OK)
    add (id, `is_last`, `sub_parties`, c, WE LOCK) to `pending`
    send (OK) to *Charlie*

// notification to hop that doesn't lock coins – doesn't ask $\mathcal{E}$

Simulator $\mathcal{S}$– Pt. 1

On $(\textsc{open}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}}, F, \text{sig}_F\, \textit{Alice})$ 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 $(\textsc{submit}, (F, \text{sig}_F))$ to $\mathcal{G}_{\text{Ledger}}$

On $(\textsc{open}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}}, pk_{B,F}, \textit{Bob})$ 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 $(\textsc{pay}, x, \textit{Dave})$ 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 $(\textsc{fund you}, c, \textit{Bob}, \textit{Charlie}, \textit{Alice})$ by $\mathcal{F}_{\text{Chan}}$:
    simulate *Alice* receiving input $(\textsc{fund you}, c, \textit{Bob})$ 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 10 of Fig. 6 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. 13.

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. 14.

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. 13. During the latter steps, $\mathcal{S}$ simulates executing line 10 of Fig. 6 with *Alice*.

Similarly in the real world, *Alice* executes lines 5 and 10 of Fig. 6, therefore the two worlds still are perfectly indistinguishable.

The "for" loop of lines 11-13 of Fig. 6 is then executed in both the real and the ideal worlds. The message of line 12 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 20 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 21-7 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:*

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

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

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

References