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.

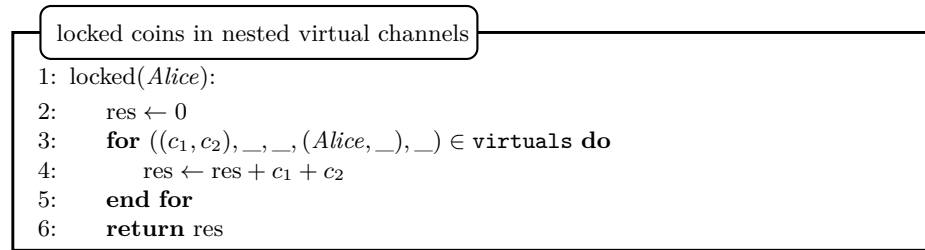The following functions are available for both Π_{Chan} and $\mathcal{F}_{\text{Chan}}$.

---

locked coins in nested virtual channels

1: locked($Alice$):
2:     res $\leftarrow$ 0
3:     **for** $((c_1, c_2), \_, \_, (Alice, \_), \_) \in$ `virtuals` **do**
4:         res $\leftarrow$ res $+ c_1 + c_2$
5:     **end for**
6:     **return** res

Fig. 1.

Protocol Π_{Chan}

1: On (INIT, out_keys) by $\mathcal{E}$:
2:      ensure $State = \bot$
3:      $(c_A, c_B) \leftarrow (0, 0)$
4:      virtuals $\leftarrow \emptyset$
5:      ensure PCN.INIT(keys, $Alice$) returns (OK)
6:      $State \leftarrow$ INIT

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

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

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

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

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

Fig. 2.

```
┌─────────────────────────────────────────────────────────────────────┐
│  ⎧ Protocol Π_Chan – virtual ⎫                                        │
│  ─────────────────────────────────────────────────────────────────   │
```

Protocol Π_{Chan} – virtual

1: // notification to fundee
2: // trust that *Charlie* has c in her channel
3: On input (OPEN VIRTUAL, c, *Bob*, host_bob) by *Charlie*:

4:     ensure $State = $ INIT
5:     ensure PCN.OPENVIRTUAL(*Bob*, *Charlie*, host_bob, c) returns (OK, keys)
6:     host_alice ← *Charlie*
7:     $c_A \leftarrow c$; $c_B \leftarrow 0$
8:     from now on, handle any (RELAYED, m) input by host_alice as the input (m) by $\mathcal{E}$
9:     from now on, transform any output (m) to $\mathcal{E}$ to output (RELAY, m) to host_alice
10:     $State \leftarrow$ OPEN VIRTUAL
11:     output (OK, keys) to *Charlie*

12: On (FUND, c, hops, inner_parties = (funder, fundee), outer_parties = (host_funder, host_fundee)) by $\mathcal{E}$:
13:     ensure $State \in \{$OPEN BASE, OPEN VIRTUAL$\}$
14:     ensure $c_A - \mathrm{locked}(A) \geq c$
15:     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*

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

18: On output (RELAY, m) by *Charlie*:
19:     do the same as in $\mathcal{F}_{\mathrm{Chan}}$, 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 $\mathcal{F}_{\text{Chan}}$ – init, top up & corruption

1: On (INIT, out_keys) by *Alice*:
2:      ensure $State \in \{\bot, \text{INIT}_{Bob}\}$
3:      $(c_A, c_B) \leftarrow (0, 0)$
4:      virtuals $\leftarrow \emptyset$
5:      ensure PCN.INIT(keys, *Alice*) returns (OK)
6:      **if** $State = \bot$ **then**
7:          $State \leftarrow \text{INIT}_{Bob}$
8:      **else** // $State = \text{INIT}_{Bob}$
9:          $State \leftarrow \text{INIT}$
10:     **end if**

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

16: On (CORRUPT) by P, addressed to *Alice*:
17:      ensure $P \in \{\text{host\_alice}, \mathcal{A}\}$
18:      virtual_secrets $\leftarrow \emptyset$
19:      **for all** $(\_, \_, (\text{fundee}, \_), (Alice, \_), vid) \in$ virtuals **do**
20:          send (CORRUPT) to fundee and ensure reply is (CORRUPTED, secrets)
21:          append (secrets, vid) to virtual_secrets
22:      **end for**
23:      from now on, allow $\mathcal{A}$ to handle all *Alice*'s messages, i.e. act as a relay
24:      **if** *Bob* is not corrupted **then**
25:          from now on, handle all messages by *Bob* as Π_{Chan} (Fig. 2-3)
26:      **end if**
27:      **if** $P = \text{host\_alice}$ **then**
28:          output (CORRUPTED, (LN.SECRETS(*Alice*), virtual_secrets)) to
     host_alice
29:      **else** // $P = \mathcal{A}$
30:          send (CORRUPTED, (LN.SECRETS(*Alice*), virtual_secrets)) to $\mathcal{A}$
31:      **end if**

Fig. 4.

4

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

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

8: On (PAY, x) by $Dave \in \{Alice, Bob\}$:
9:      ensure $State \in \{$OPEN BASE, OPEN VIRTUAL$\}$
10:      ensure $c_D - \text{locked}(D) \geq x$
11:      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
12:      $c_D \leftarrow c_D - x$; $c_{\bar{D}} \leftarrow c_{\bar{D}} + x$ // $\bar{D}$ is *Alice* if D is *Bob* and vice-versa
13:      output (PAY SUCCESS) to *Dave*

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

17: On (CLOSE) by *Alice*:
18:      ensure $State \in \{$OPEN BASE, OPEN VIRTUAL$\}$
19:      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
20:      **if** $State = $ OPEN BASE **then**
21:          ensure PCN.CLOSE(*Alice*, $(\text{tx}_i, (\sigma_{ij})_j)_i$) returns (OK)
22:          $State \leftarrow$ CLOSED
23:          output (CLOSE SUCCESS) to *Alice*
24:      **else** // $State = $ OPEN VIRTUAL
25:          $State \leftarrow$ CLOSED
26:          output (CLOSED VIRTUAL, $(\text{tx}_i, (\sigma_{ij})_j)_i$) to `host_alice` as *Alice*
27:      **end if**

28: On ((PEER) CLOSED VIRTUAL, $(\text{tx}_i, (\sigma_{ij})_j)_i$) by *Charlie*:
29:      ensure $State \in \{$OPEN BASE, OPEN VIRTUAL$\}$
30:      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 $= pk_{A,V}, pk_{B,V}$
31:      ensure VIRT.CLOSED(c_L, c_R, $(\text{tx}_i, (\sigma_{ij})_j)_i$) returns (OK)
32:      add message contents to `virtuals` entry
33:      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

1: On (FUND, c, hops, inner_parties = (funder, fundee), outer_parties = (host_funder, host_fundee)) by *Alice*: // we fund another channel

2:     ensure $State \in \{\text{OPEN BASE}, \text{OPEN VIRTUAL}\}$

3:     ensure $c_A - \text{locked}(A) \geq c$

4:     ensure host_funder = *Alice*

5:     generate unique *vid*

6:     ensure VIRT.FUND(c, hops, inner_parties, outer_parties, PCN, *vid*) returns (OK)

7:     add $((c, 0)$, hops, inner_parties, outer_parties, *vid*) to virtuals

8:     output (FUND SUCCESS) to *Alice*

9: On input (OPEN VIRTUAL, c, fundee, host_fundee) by host_funder to *Alice*: // *Alice* is funded by host_funder

10:     ensure $State = \text{INIT}$

11:     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

12:     input (YOU ARE HOST, c, funder, fundee, host_funder) to host_fundee

13:     ensure $c_A \geq c$

14:     ensure we are not already supporting a virtual channel

15:     output (OK)

16:     ensure PCN.OPENVIRTUAL(fundee, host_funder, host_fundee, c) returns (OK, keys)

17:     $c_A \leftarrow c; c_B \leftarrow 0$

18:     from now on, handle any (RELAYED, m) input by {host_funder, host_fundee} as if it were input (m) by {*Alice*, *Bob*} respectively

19:     from now on, transform any output (m) to {*Alice*, *Bob*} to output (RELAY, m) to {host_funder, host_fundee} respectively

20:     $State \leftarrow \text{OPEN VIRTUAL}$

21:     output (OK, keys) to host_funder

22: On (RELAY, m, *Charlie*) by *Alice*:

23:     ensure there is an entry in virtuals with *Alice* as host of funder and *Charlie* as fundee sub-party

24:     input (RELAYED, m) to *Charlie*

25: On output (RELAY, m) by *Charlie* to *Alice*:

26:     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

27:     output (RELAYED, m, *Charlie*) to $\mathcal{E}$

Fig. 6.

6

Process LN – init

1: INIT(**keys**, *Dave*):
2:     ensure *Dave* = *Alice*
3:     $pk_{A,\text{out}} \leftarrow$ **keys**
4:     **return** (OK)

5: TOPUP(**funder**):
6:     ensure super.*State* = INIT
7:     $(sk_{\text{chain}}, pk_{\text{chain}}) \leftarrow$ KEYGEN()
8:     output (PUBLIC KEY, pk_{chain}) to **funder**
9:     **while** $\nexists \text{tx} \in \Sigma, c_{\text{chain}} : (c_{\text{chain}}, pk_{\text{chain}}) \in \text{tx.outputs}$ **do**
10:       wait[a] for input (CHECK TOP UP) by **funder**
11:       input (READ) to $\mathcal{G}_{\text{Ledger}}$ as **funder** and assign ouput to Σ
12:     **end while**
13:     `base_output` $\leftarrow (c_{\text{chain}}, pk_{\text{chain}})$
14:     **return** (OK, c_{chain})

15: // used by VIRT
16: GETCOMMTX(c_A, c_B, $pk_{A,F}$, $pk_{A,\text{out}}$, $pk_{A,R}$, $pk_{B,F}$, $pk_{B,\text{out}}$, $pk_{B,R}$, P):
17:     $C_{P,i+1} \leftarrow$ TX {input: $(c_A + c_B, 2/\{pk_{A,F}, pk_{B,F}\})$, outputs: $((c_A,$
    $(pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\}), (c_B, pk_{B,\text{out}}))$}
18:     **return** $C_{P,i+1}$

19: // used by VIRT
20: GETCOMMKEYS():
21:     **return** $(pk_{A,F}, pk_{A,\text{out}}, pk_{A,R}), (pk_{B,F}, pk_{B,\text{out}}, pk_{B,R})$

22: // used by VIRT
23: REVOKEPREVIOUS():
24:     $R_{B,i} \leftarrow$ TX {input: $C_{A,i}.\text{outputs}.Alice$, output: $(C_{A,i}.\text{outputs}.Alice.\text{value},$
    $pk_{B,\text{out}})$}
25:     $\text{sig}_{A,R,i} \leftarrow$ SIGN($R_{B,i}, sk_{A,R}$)
26:     **return** $\text{sig}_{A,R,i}$

27: // used by VIRT
28: PROCESSREMOTEREOVCATION($\text{sig}_{B,R,i}$):
29:     $R_{A,i} \leftarrow$ TX {input: $C_{B,i}.\text{outputs}.Bob$, output: $(C_{B,i}.\text{outputs}.Bob.\text{value},$
    $pk_{A,\text{out}})$}
30:     ensure VERIFY($R_{A,i}, \text{sig}_{B,R,i}, pk_{B,R}$) = True
31:     **return** (OK)

---

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

Fig. 7.

Process LN − base

1: OPENBASE(**fundee**):
2:     ensure super.$State = $ TOPPED UP
3:     $(sk_{A,F}, pk_{A,F}) \leftarrow$ KEYGEN(); $(sk_{A,R}, pk_{A,R}) \leftarrow$ KEYGEN()
4:     **if** ideal world **then**
5:         $(sk_{B,F}, pk_{B,F}) \leftarrow$ KEYGEN(); $(sk_{B,R}, pk_{B,R}) \leftarrow$ KEYGEN()
6:     **else** // real world
7:         send (OPEN BASE CHANNEL, c_{chain}, $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$) to **fundee**
8:         // colored code is run by **fundee**. Validation is implicit
9:         ensure super.$State = $ INIT // "super": storage of enclosing protocol
10:         store $pk_{A,F}$, $pk_{A,R}$, $pk_{A,\text{out}}$
11:         $(sk_{B,F}, pk_{B,F}) \leftarrow$ KEYGEN(); $(sk_{B,R}, pk_{B,R}) \leftarrow$ KEYGEN()
12:         reply (ACCEPT BASE CHANNEL, $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$)
13:         store $pk_{B,F}$, $pk_{B,R}$, $pk_{B,\text{out}}$
14:     **end if**
15:     $F \leftarrow$ TX {input: `base_output`, output: $(c_{\text{chain}}, 2/\{pk_{A,F}, pk_{B,F}\})$}
16:     **if** real world **then**
17:         $C_{A,0} \leftarrow$ TX {input: F.output, outputs: $(c_{\text{chain}}, (pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}
18:         $C_{B,0} \leftarrow$ TX {input: F.output, outputs: $(c_{\text{chain}}, pk_{A,\text{out}})$, $(0, (pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}
19:         $\text{sig}_{A,C,0} \leftarrow$ SIGN($C_{B,0}$, $sk_{A,F}$)
20:         send (FUNDING CREATED, `base_output`, $\text{sig}_{A,C,0}$) to **fundee**
21:         // implicitly verify that this is a continuation of the previous exchange
22:         $F \leftarrow$ TX {input: `base_output`, output: $(c_{\text{chain}}, 2/\{pk_{A,F}, pk_{B,F}\})$}
23:         $C_{B,0} \leftarrow$ TX {input: F.output, outputs: $(c_{\text{chain}}, pk_{A,\text{out}})$, $(0, (pk_{B,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})$}
24:         ensure VERIFY($C_{B,0}$, $\text{sig}_{A,C,0}$, $pk_{A,F}$) = True
25:         $C_{A,0} \leftarrow$ TX {input: F.output, outputs: $(c_{\text{chain}}, (pk_{A,\text{out}} + t) \vee 2/\{pk_{A,R}, pk_{B,R}\})$, $(0, pk_{B,\text{out}})$}
26:         $\text{sig}_{B,C,0} \leftarrow$ SIGN($C_{A,0}$, $sk_{B,F}$)
27:         reply (FUNDING SIGNED, $\text{sig}_{B,C,0}$)
28:         ensure VERIFY($C_{A,0}$, $\text{sig}_{B,C,0}$, $pk_{B,F}$) = True
29:     **end if**
30:     $\text{sig}_F \leftarrow$ SIGN(F, sk_{chain})
31:     send (OPEN, c_{chain}, $pk_{A,\text{out}}$, $pk_{B,\text{out}}$, F, sig_F, **funder**) to $\mathcal{A}$
32:     **while** $F \notin \Sigma$ **do**
33:         wait for input (CHECK FUNDING) by **funder**
34:         input (READ) to $\mathcal{G}_{\text{Ledger}}$ as **funder** and assign output to Σ
35:     **end while**
36:     **return** (OK, c_{chain})

Fig. 8.

8

Process LN – open virtual

1: OPENVIRTUAL(`fundee`, `host_funder`, `host_fundee`, c):
2:     **if** real world **then**
3:         do funding ceremony as in base channel (Fig. 8, lines 15-30) TODO: abstract better
4:             **return** $(pk_{A,F}, pk_{B,F})$
5:     **end if**
6:     **return** (OK)

Fig. 9.

Process LN – pay

1: PAY(x, *payid*): // *Alice* pays, *Bob* gets paid
2:     $C_{B,i+1} \leftarrow C_{B,i}$ with x coins moved from *Alice*'s to *Bob*'s output
3:     $\text{sig}_{A,C,i+1} \leftarrow \text{SIGN}(C_{B,i+1}, sk_{A,F})$ // kept by *Alice*
4:     send (PAY, x, $\text{sig}_{A,C,i+1}$, *payid*) to *Bob*
5:     $C_{B,i+1} \leftarrow C_{B,i}$ with x coins moved from *Alice*'s to *Bob*'s output
6:     ensure VERIFY($C_{B,i+1}$, $\text{sig}_{A,C,i+1}$, $pk_{A,F}$) = True
7:     $C_{A,i+1} \leftarrow C_{A,i}$ with x coins moved from *Alice*'s to *Bob*'s output
8:     $\text{sig}_{B,C,i+1} \leftarrow \text{SIGN}(C_{A,i+1}, sk_{B,F})$ // kept by *Bob*
9:     $R_{A,i} \leftarrow$ TX {input: $C_{B,i}$.outputs.*Alice*, output: $(c_B, pk_{A,\text{out}})$}
10:     $\text{sig}_{B,R,i} \leftarrow \text{SIGN}(R_{A,i}, sk_{B,R})$
11:     reply (COMMITMENT SIGNED, $\text{sig}_{B,C,i+1}$, $\text{sig}_{B,R,i}$)
12:     $C_{A,i+1} \leftarrow C_{A,i}$ with x coins moved from *Alice*'s to *Bob*'s output
13:     ensure VERIFY($C_{A,i+1}$, $\text{sig}_{B,C,i+1}$, $pk_{B,F}$) = True
14:     $R_{A,i} \leftarrow$ TX {input: $C_{B,i}$.outputs.*Alice*, output: $(c_B, pk_{A,\text{out}})$}
15:     ensure VERIFY($R_{A,i}$, $\text{sig}_{B,R,i}$, $pk_{B,R}$) = True
16:     $R_{B,i} \leftarrow$ TX {input: $C_{A,i}$.outputs.*Bob*, output: $(c_A, pk_{B,\text{out}})$}
17:     $\text{sig}_{A,R,i} \leftarrow \text{SIGN}(R_{B,i}, sk_{A,R})$
18:     add (x, *payid*) to `paid_out`
19:     $c_A \leftarrow c_A - x; c_B \leftarrow c_B + x$
20:     send (REVOKE AND ACK, $\text{sig}_{A,R,i}$) to *Bob*
21:     $R_{B,i} \leftarrow$ TX {input: $C_{A,i}$.outputs.*Bob*, output: $(c_A, pk_{B,\text{out}})$}
22:     ensure VERIFY($R_{B,i}$, $\text{sig}_{A,R,i}$, $pk_{A,R}$) = True
23:     add (x, *payid*) to `paid_in`
24:     $c_A \leftarrow c_A - x; c_B \leftarrow c_B + x$

Fig. 10.

Process LN – close

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

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

Fig. 11.

Process VIRT

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

10: CIRCULATEVIRTUALKEYS(hops, $(pk_{A,V}, pk_{B,V})$, left_virt_keys):
11:     // hops is a list of pid pairs
12:     $(sk_{\mathrm{loc}}, pk_{\mathrm{loc}}) \leftarrow$ KEYGEN()
13:     **if** left_virt_keys is given as argument **then** // we are not host_funder
14:         **if** len(hops[1:]) ≥ 1 **then** // we are not host_fundee
15:             input (VIRTUAL KEYS FORWARD, hops[1:], $(pk_{A,V}, pk_{B,V})$, left_virt_keys, $(sk_{\mathrm{loc}}, pk_{\mathrm{loc}})$, $(pk_{A,F}, pk_{B,F})$) to hops[1].$Alice$ // sibling
16:             store inputs as left_virt_keys, $(sk_{\mathrm{loc}}, pk_{\mathrm{loc}})$, left_fund_keys
17:             call VIRT.CIRCULATEVIRTUALKEYS(hops, $(pk_{A,V}, pk_{B,V})$, $(pk_{\mathrm{loc}}$, left_virt_keys[0])) of hops[0].Bob and assign output to right_virt_keys, right_fund_keys
18:             output (VIRTUAL KEYS BACK, right_virt_keys, $(pk_{A,F}, pk_{B,F})$)
19:             store outputs as right_virt_keys, right_fund_keys
20:             **return** $(pk_{\mathrm{loc}}$, right_virt_keys[0]), right_fund_keys
21:         **else** // we are host_fundee
22:             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.
23:             call PCN.GETCOMMKEYS() of fundee and assign output to $(pk'_{A,V}, \_, \_)$, $(pk'_{B,V}, \_, \_)$
24:             ensure $pk_{A,V} = pk'_{A,V} \wedge pk_{B,V} = pk'_{B,V}$
25:             **return** pk_{loc}
26:         **end if**
27:     **else** // we are host_funder
28:         call VIRT.CIRCULATEVIRTUALKEYS(hops, $(pk_{A,V}, pk_{B,V})$, pk_{loc}) of hops[0].Bob and assign output to $(pk_{\mathrm{right}}, pk_{\mathrm{right},2})$, right_fund_keys
29:         **return** (OK)
30:     **end if**

Fig. 12.

11

Process VIRT

1: 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`, P):

2:   ensure $c_{\text{sibRem}} \geq c_{\text{guest}} \wedge c_{\text{loc}} \geq c_{\text{guest}}$

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

4:   `left_fund` $\leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{loc,fund}}\}$

5:   `right_fund` $\leftarrow 2/\{pk_{\text{sib,fund}}, pk_{\text{right,fund}}\}$

6:   `left_virt` $\leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{loc,virt}}\}$

7:   `right_virt` $\leftarrow 2/\{pk_{\text{sib,virt}}, pk_{\text{right,virt}}\}$

8:   `guest` $\leftarrow 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$

9:   `left_out` $\leftarrow (2 \wedge$ `left_virt`$) \vee (3 \wedge$ `left_virt` $+ t)$

10:   `right_out` $\leftarrow (1 \wedge$ `right_virt`$) \vee (3 \wedge$ `right_virt` $+ t)$

11:   `guest_out` $\leftarrow$ `guest_virt` $\vee ($`guest_virt` $+ t)$

12:   $\text{TX}_{\text{none}} \leftarrow$ TX {inputs: $((c_{\text{left}}, $ `left_fund`$), (c_{\text{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_{\text{guest}}, $ `guest_out`$))$}

13:   $\text{TX}_{\text{left}} \leftarrow$ TX {inputs: $((c_{\text{left}} - c_{\text{guest}}, 1 \wedge$ `left_virt`$), (c_{\text{right}}, $ `right_fund`$))$, outputs: $((c_{\text{left}} - c_{\text{guest}}, 3 \wedge$ `left_virt`$), (c_{\text{right}} - c_{\text{guest}}, $ `right_out`$), (c_{\text{guest}}, pk_{\text{loc,out}}))$}

14:   $\text{TX}_{\text{right}} \leftarrow$ TX {inputs: $((c_{\text{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`$), (c_{\text{guest}}, pk_{\text{loc,out}}))$}

15:   $\text{TX}_{\text{both}} \leftarrow$ TX {inputs: $((c_{\text{left}} - c_{\text{guest}}, 1 \wedge$ `left_virt`$), (c_{\text{right}} - c_{\text{guest}}, 2 \wedge$ `right_virt`$), (c_{\text{guest}}, $ `guest`$))$, outputs: $((c_{\text{left}} - c_{\text{guest}}, 3 \wedge$ `left_virt`$), (c_{\text{right}} - c_{\text{guest}}, 3 \wedge$ `right_virt`$), (c_{\text{guest}}, pk_{\text{loc,out}}))$}

16:
   $C \leftarrow$ PCN.GETCOMMTX($c_{\text{loc}} - c_{\text{guest}}, c_{\text{rem}}$, `comm_keys_loc`, `comm_keys_rem`, P)

17:   **return** (TX_{none}, TX_{left}, TX_{right}, TX_{both}, C)

Fig. 13.

12

Process VIRT

1: // left and right refer to the two counterparties, with left being the one closer to the funder. Note difference with left/right meaning in VIRT.GETMIDTXS.
2: GETEDGETXS(c_{guest}, c_{left}, c_{right}, $pk_{\text{left,fund}}$, $pk_{\text{right,fund}}$, $pk_{\text{left,virt}}$, $pk_{\text{right,virt}}$, $pk_{\text{left,guest}}$, $pk_{\text{right,guest}}$, comm_keys_left, comm_keys_right, is_funder, P):
3:     ensure $c_{\text{left}} \geq c_{\text{guest}}$
4:     $c_{\text{tot}} \leftarrow c_{\text{left}} + c_{\text{right}}$
5:     $\texttt{fund} \leftarrow 2/\{pk_{\text{left,fund}}, pk_{\text{right,fund}}\}$
6:     $\texttt{virt} \leftarrow 2/\{pk_{\text{left,virt}}, pk_{\text{right,virt}}\}$
7:     $\texttt{guest} \leftarrow 2/\{pk_{\text{left,guest}}, pk_{\text{right,guest}}\}$
8:     **if** is_funder = True **then**
9:         $\texttt{out} \leftarrow (1 \wedge \texttt{virt}) \vee (3 \wedge \texttt{virt} + t)$
10:     **else** // TXs belong to fundee
11:         $\texttt{out} \leftarrow (2 \wedge \texttt{virt}) \vee (3 \wedge \texttt{virt} + t)$
12:     **end if**
13:     $\texttt{guest\_out} \leftarrow \texttt{guest\_virt} \vee (\texttt{guest\_virt} + t)$
14:     $\text{TX}_{\text{base}} \leftarrow$ TX {input: c_{tot}, fund, outputs: $((c_{\text{tot}} - c_{\text{guest}}, \texttt{out}), (c_{\text{guest}}, \texttt{guest\_out}))$}
15:     $C \leftarrow$ PCN.GETCOMMTX($c_{\text{left}} - c_{\text{guest}}$, c_{right}, comm_keys_left, comm_keys_right, P)
16:     **return** $(\text{TX}_{\text{base}}, C)$

Fig. 14.

13

Process VIRT.SIBLINGSIGS()

1: $(\texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}) \leftarrow$ PCN.GETCOMMKEYS()
2: $\text{sigs}_{\text{toRight}} \leftarrow \emptyset$
3: **if** we have $\texttt{right\_virt\_keys}[1]$ **then** // next hop is not host_fundee
4:     $(\text{TX}_{\text{next,none}}, \text{TX}_{\text{next,left}}, \text{TX}_{\text{next,right}}, \text{TX}_{\text{next,both}}, C) \leftarrow$
   VIRT.GETMIDTXS($c, c_A, c_B, pk_{A,F}, pk_{B,F}, \texttt{right\_fund\_keys}, pk_{loc},$
   $\texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[1], pk_{A,V}, pk_{B,V},$
   $\texttt{right\_virt\_keys}[0], \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, B)$
5:     **for** TX in $(\text{TX}_{\text{next,left}}, \text{TX}_{\text{next,right}}, \text{TX}_{\text{next,both}}, C)$ **do**
6:         add SIGN(TX, sk_{loc}, ANYPREVOUT) to $\text{sigs}_{\text{toRight}}$
7:     **end for**
8: **else** // next hop is host_fundee
9:     $(\text{TX}_{\text{next,none}}, C) \leftarrow$ VIRT.GETEDGETXS($c, c_A, c_B, pk_{A,F}, pk_{B,F}, pk_{loc},$
   $\texttt{right\_fund\_keys}[0], pk_{A,V}, pk_{B,V}, \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, \text{False}, B)$
10:     add SIGN(C, sk_{loc}, ANYPREVOUT) to $\text{sigs}_{\text{toRight}}$
11: **end if**
12: call VIRT.CIRCULATEVIRTUALSIGS(c, $\texttt{hops}$, $\text{sigs}_{\text{toRight}}$) of $\texttt{hops}[0].Bob$ and
   assign output to $\text{sigs}_{\text{byRight}}$
13: $(\text{TX}_{\text{loc,none}}, \text{TX}_{\text{loc,left}}, \text{TX}_{\text{loc,right}}, \text{TX}_{\text{loc,both}}, C) \leftarrow$ VIRT.GETMIDTXS($c, c_A,$
   $c_B, \texttt{left\_fund\_keys}, pk_{A,F}, pk_{B,F}, \texttt{left\_virt\_keys}[0], pk_{loc}, pk_{loc},$
   $\texttt{right\_virt\_keys}[0], pk_{A,V}, pk_{B,V}, pk_{loc}, \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, A)$
14: **for** TX, i in $(\text{TX}_{\text{loc,left}}, \text{TX}_{\text{loc,right}}, \text{TX}_{\text{loc,both}}, C)$ **do**
15:     ensure VERIFY(TX, $\text{sigs}_{\text{byRight}}[i]$, $\texttt{right\_virt\_keys}[0]$) = True
16: **end for**
17: output (VIRTUALSIGSBACK, $\text{sigs}_{\text{byRight}}$) // sibling needs $\text{sigs}_{\text{byRight}}$ for closing

Fig. 15.

14

Process VIRT.INTERMEDIARYSIGS()

1: $(TX_{loc,none}, TX_{loc,left}, TX_{loc,right}, TX_{loc,both}, C) \leftarrow$ VIRT.GETMIDTXS$(c, c_A,$ $c_B, pk_{A,F}, pk_{B,F}, \texttt{right\_fund\_keys}, \texttt{left\_virt\_keys}[0], pk_{loc}, pk_{loc},$ $\texttt{right\_virt\_keys}[0], pk_{A,V}, pk_{B,V}, pk_{loc}, \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, A)$
2: **for** TX, i in $(TX_{loc,left}, TX_{loc,right}, TX_{loc,both}, C)$ **do**
3:     ensure VERIFY$(TX, \text{sigs}_{byLeft}[i], \texttt{left_virt_keys}[0]) = $ True
4: **end for**
5: input (VIRTUAL SIGS FORWARD, $\texttt{hops}[1:], \text{sigs}_{byLeft}$) to $\texttt{hops}[1].Alice$ // sibling needs sigs$_{byLeft}$ for closing
6: VIRT.SIBLINGSIGS()
7: $\text{sigs}_{toLeft} \leftarrow \emptyset$
8: **if** we have $\texttt{left\_virt\_keys}[1]$ **then** // previous hop is not host_funder
9:     $(TX_{prev,none}, TX_{prev,left}, TX_{prev,right}, TX_{prev,both}, C) \leftarrow$ VIRT.GETMIDTXS$(c, c_A, c_B, pk_{A,F}, pk_{B,F}, \texttt{right\_fund\_keys}, pk_{loc},$ $\texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[1], pk_{A,V}, pk_{B,V},$ $\texttt{right\_virt\_keys}[0], \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, B)$
10:     **for** TX in $(TX_{prev,left}, TX_{prev,right}, TX_{prev,both}, C)$ **do**
11:         add SIGN$(TX, sk_{loc}, \text{ANYPREVOUT})$ to sigs_{toLeft}
12:     **end for**
13: **else** // previous hop is host_funder
14:     $(TX_{prev,none}, C) \leftarrow$ VIRT.GETEDGETXS$(c, c_B, c_A, pk_{B,F}, pk_{A,F},$ $\texttt{left\_virt\_keys}[0], pk_{loc}, pk_{A,V}, pk_{B,V}, \texttt{comm\_keys\_rem}, \texttt{comm\_keys\_loc}, \text{True},$ $B)$
15:     add SIGN$(C, sk_{loc}, \text{ANYPREVOUT})$ to sigs_{toLeft}
16: **end if**
17: **return** sigs_{toLeft}

Fig. 16.

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

1: $(\texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}) \leftarrow \text{PCN.GETCOMMKEYS}()$
2: **if** $\text{sigs}_{\text{byLeft}}$ is given as argument **then** // we are not host_funder
3:     **if** $\text{len}(\texttt{hops}[1:]) \geq 1$ **then** // we are not host_fundee
4:         **return** VIRT.INTERMEDIARYSIGS()
5:     **else** // we are host_fundee
6:         $(\text{TX}_{\text{loc,none}}, C) \leftarrow \text{VIRT.GETEDGETXS}(c, c_B, c_A, pk_{B,F}, pk_{A,F},$ $\texttt{left\_virt\_keys}[0], pk_{loc}, pk_{A,V}, pk_{B,V}, \texttt{comm\_keys\_rem}, \texttt{comm\_keys\_loc},$ False, $A)$
7:         ensure $\text{VERIFY}(C, \text{sigs}_{\text{byLeft}}[0], \texttt{left\_virt\_keys}[0]) = \text{True}$
8:         $(\text{TX}_{\text{prev,none}}, \text{TX}_{\text{prev,left}}, \text{TX}_{\text{prev,right}}, \text{TX}_{\text{prev,both}}, C) \leftarrow$ $\text{VIRT.GETMIDTXS}(c, c_A, c_B, pk_{A,F}, pk_{B,F}, \texttt{right\_fund\_keys}, pk_{loc},$ $\texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[1], pk_{A,V}, pk_{B,V},$ $\texttt{right\_virt\_keys}[0], \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, B)$
9:         $\text{sigs}_{\text{toLeft}} \leftarrow \emptyset$
10:         **for** TX in $(\text{TX}_{\text{prev,left}}, \text{TX}_{\text{prev,right}}, \text{TX}_{\text{prev,both}}, C)$ **do**
11:             add $\text{SIGN}(\text{TX}, sk_{\text{loc}}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toLeft}}$
12:         **end for**
13:         **return** $\text{sigs}_{\text{toLeft}}$
14:     **end if**
15: **else** // we are host_funder
16:     $(\text{TX}_{\text{next,none}}, \text{TX}_{\text{next,left}}, \text{TX}_{\text{next,right}}, \text{TX}_{\text{next,both}}, C) \leftarrow$ $\text{VIRT.GETMIDTXS}(c, c_A, c_B, pk_{A,F}, pk_{B,F}, \texttt{right\_fund\_keys}, pk_{loc},$ $\texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[0], \texttt{right\_virt\_keys}[1], pk_{A,V}, pk_{B,V},$ $\texttt{right\_virt\_keys}[0], \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, B)$
17:     $\text{sigs}_{\text{toRight}} \leftarrow \emptyset$
18:     **for** TX in $(\text{TX}_{\text{next,left}}, \text{TX}_{\text{next,right}}, \text{TX}_{\text{next,both}}, C)$ **do**
19:         add $\text{SIGN}(\text{TX}, sk_{\text{loc}}, \text{ANYPREVOUT})$ to $\text{sigs}_{\text{toRight}}$
20:     **end for**
21:     call VIRT.CIRCULATEVIRTUALSIGS(c, hops, $\text{sigs}_{\text{toRight}}$) of $\texttt{hops}[0].Bob$ and assign output to $\text{sigs}_{\text{byRight}}$
22:     $(\text{TX}_{\text{loc,none}}, C) \leftarrow \text{VIRT.GETEDGETXS}(c, c_A, c_B, pk_{A,F}, pk_{B,F}, pk_{loc},$ $\texttt{right\_virt\_keys}[0], pk_{A,V}, pk_{B,V}, \texttt{comm\_keys\_loc}, \texttt{comm\_keys\_rem}, \text{True}, A)$
23:     ensure $\text{VERIFY}(C, \text{sigs}_{\text{byRight}}[0], \texttt{right\_virt\_keys}[0]) = \text{True}$
24:     **return** (OK)
25: **end if**

Fig. 17.

16

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

1: **if** $\text{sig}_{\text{loc,none}}$ is given as argument **then** // we are not host_funder
2:     ensure VERIFY($\text{TX}_{\text{loc,none}}$, $\text{sig}_{\text{loc,none}}$, left_virt_keys[0]) = True
3:     **if** len(hops[1:]) ≥ 1 **then** // we are not host_fundee
4:         input (VIRTUAL BASE SIG FORWARD, hops[1:], $\text{sig}_{\text{loc,none}}$) to hops[1].*Alice* // sibling needs $\text{sig}_{\text{loc,none}}$ for closing
5:         $\text{sig}_{\text{next,none}} \leftarrow$ SIGN($\text{TX}_{\text{next,none}}$, $sk_{\text{loc},F}$)
6:         call VIRT.CIRCULATEVIRTUALSIGS(hops, $\text{sig}_{\text{next,none}}$) of hops[0].*Bob* and assign output to $\text{sig}_{\text{loc,none}}$
7:         ensure VERIFY($\text{TX}_{\text{loc,none}}$, $\text{sig}_{\text{loc,none}}$, right_virt_keys[0]) = True
8:         output (VIRTUAL BASE SIG BACK, $\text{sig}_{\text{loc,none}}$) // sibling needs $\text{sig}_{\text{loc,none}}$ for closing
9:     **end if**
10:     $\text{sig}_{\text{prev,none}} \leftarrow$ SIGN($\text{TX}_{\text{prev,none}}$, $sk_{\text{loc},F}$)
11:     **return** $\text{sig}_{\text{prev,none}}$
12: **else** // we are host_funder
13:     $\text{sig}_{\text{next,none}} \leftarrow$ SIGN($\text{TX}_{\text{next,none}}$, $sk_{\text{loc},F}$)
14:     call VIRT.CIRCULATEFUNDINGSIGS(hops, $\text{sig}_{\text{next,none}}$) of hops[0].*Bob* and assign output to $\text{sig}_{\text{loc,none}}$
15:     ensure VERIFY($\text{TX}_{\text{loc,none}}$, $\text{sig}_{\text{loc,none}}$, right_virt_keys[0]) = True
16:     **return** (OK)
17: **end if**

Fig. 18.

17

Process VIRT.CIRCULATEREVOCATIONS(hops, revoc_by_prev)

1: **if** revoc_by_prev is given as argument **then** // we are not host_funder
2:     ensure PCN.PROCESSREMOTEREVOCATION(revoc_by_prev) returns (OK)
3: **else** // we are host_funder
4:     revoc_for_next ← PCN.REVOKEPREVIOUS()
5:     call VIRT.CIRCULATEREVOCATIONS(hops, revoc_for_next) of hops$[0].Bob$
   and assign output to revoc_by_next
6:     ensure PCN.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK)
7:     **return** (OK)
8: **end if**
9: **if** len(hops$[1:]$) $\geq$ 1 **then** // we are not host_fundee nor host_funder
10:     input (VIRTUAL REVOCATION FORWARD, hops$[1:]$) to hops$[1].Alice$
11:     revoc_for_next ← PCN.REVOKEPREVIOUS()
12:     call VIRT.CIRCULATEREVOCATIONS(hops, revoc_for_next) of hops$[0].Bob$
   and assign output to revoc_by_next
13:     ensure PCN.PROCESSREMOTEREVOCATION(revoc_by_next) returns (OK)
14:     output (VIRTUAL REVOCATION BACK)
15: **end if**
16: revoc_for_prev ← PCN.REVOKEPREVIOUS()
17: **return** revoc_for_prev

Fig. 19.

18

Process VIRT

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

2: **if** both channel parties are honest **then**

3:     **if** funded $\neq \emptyset$ **then**

4:         TODO: prepare virtual layer TX V and its signature – careful, may be unneded!

5:         $C \leftarrow$ TX {input: V.output, outputs: $(c_A, pk_{A,\text{out}} + t), (c_B, pk_{B,\text{out}})$}

6:         $\text{sig}_{B,C} \leftarrow \text{SIGN}(C, sk_{B,V})$

7:         $\text{sig}_{A,C} \leftarrow \text{SIGN}(C, sk_{A,V})$

8:     **else**

9:         $C \leftarrow$ TX {input: F.output, outputs: $(c_A, pk_{A,\text{out}} + t), (c_B, pk_{B,\text{out}})$}

10:         $\text{sig}_{B,C} \leftarrow \text{SIGN}(C, sk_{B,F})$

11:         $\text{sig}_{A,C} \leftarrow \text{SIGN}(C, sk_{A,F})$

12:     **end if**

13: **end if**   // if *Bob* is corrupted, we already have C and $\text{sig}_{B,C}$

14: $State \leftarrow$ CLOSED

15: **if** $State =$ OPEN BASE **then**

16:     input (SUBMIT, $(V, \text{sig}_{A,V}, \text{sig}_{B,V}), (C, \text{sig}_{A,C}, \text{sig}_{B,C})$) to $\mathcal{G}_{\text{Ledger}}$

17: **else**   // $State =$ OPEN VIRTUAL

18:     **if** *Alice* is the one that had received (OPEN VIRTUAL, . . .) by **opener then**

19:         initiator $\leftarrow$ opener; other $\leftarrow$ outer_peer

20:     **else**   // *Bob* had received (OPEN VIRTUAL . . .)

21:         initiator $\leftarrow$ outer_peer; other $\leftarrow$ opener

22:     **end if**

23:     **if** both parties are honest **then**

24:         **if** funded $\neq \emptyset$ **then**

25:             TODO: prepare virtual layer TXs V_A, V_B and their signatures

26:             $C' \leftarrow$ TX {input: V.output, outputs: $(c_A, pk_{A,\text{out}}), (c_B, pk_{B,\text{out}} + t)$}

27:             $\text{sig}'_{A,C} \leftarrow \text{SIGN}(C', sk_{A,V}); \text{sig}'_{B,C} \leftarrow \text{SIGN}(C', sk_{B,V})$

28:         **else**   // there are no virtual channels on top of us

29:             $C' \leftarrow$ TX {input: F.output, outputs: $(c_A, pk_{A,\text{out}}), (c_B, pk_{B,\text{out}} + t)$}

30:             $\text{sig}'_{A,C} \leftarrow \text{SIGN}(C', sk_{A,F}); \text{sig}'_{B,C} \leftarrow \text{SIGN}(C', sk_{B,F})$

31:         **end if**

32:         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*

33:     **end if**

34:     **if** funded $\neq \emptyset$ **then**

35:         output $\leftarrow$ (CLOSED VIRTUAL, $(V, \text{sig}_{A,V}, \text{sig}_{B,V}), (C, \text{sig}_{A,C}, \text{sig}_{B,C})$)

36:     **else**

37:         output $\leftarrow$ (CLOSED VIRTUAL, $(C, \text{sig}_{A,C}, \text{sig}_{B,C})$)

38:     **end if**

39:     output output to initiator as *Alice*

40: **end if**

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

42:     **for all** i in $1 \ldots |(\text{tx}_i, (\sigma_{ij})_j)_i|$ **do**

43:         ensure VERIFY($\text{tx}_i, (\sigma_{ij})_j$) = True

44:     **end for**

45:     ensure $(\text{tx}_1, (\sigma_{1j})_j)$ has exactly 1 input, which spends an output of V of value $c_L + c_R$

46:     return (OK)

19

Fig. 20.

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

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

6: On (IS PARENT, id_1, id_2) by P
7:     **if** id_1 is the parent of id_2 in G **then**
8:         send (IS PARENT, id_1, id_2, true) to P
9:     **else**
10:        send (IS PARENT, id_1, id_2, false) to P
11:     **end if**

Simulator $\mathcal{S}$– Pt. 1

1: On $(\text{OPEN}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}}, F, \text{sig}_F \, Alice)$ by $\mathcal{F}_{\text{Chan}}$: // both honest

2:      simulate *Alice* receiving input $(\text{OPEN}, c_F, pk_{A,\text{out}}, pk_{B,\text{out}})$ by $\mathcal{E}$

3:      ensure simulated *Alice* inputs $(\text{SUBMIT}, (F', \text{sig}_{F'}))$ to $\mathcal{G}_{\text{Ledger}}$

4:      input $(\text{SUBMIT}, (F, \text{sig}_F))$ to $\mathcal{G}_{\text{Ledger}}$

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

6:      send LN message $(\text{OPEN}, pk_{B,F})$ 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

7: On $(\text{PAY}, x, Dave)$ by $\mathcal{F}_{\text{Chan}}$:

8:      **if** both channel parties are honest **then**

9:          simulate *Dave* receiving input (PAY, x) by $\mathcal{E}$

10:          ensure simulated *Dave* outputs (OK)

11:          send (OK) to $\mathcal{F}_{\text{Chan}}$

12:      **else if** only *Dave*'s counterparty is corrupted **then** // else just relay to $\mathcal{A}$

13:          simulate *Dave* receiving input (PAY, x) by $\mathcal{E}$

14:          ensure simulated *Dave* outputs (OK)

15:          extract the latest commitment transaction C and its signature by *Dave*'s counterparty $\text{sig}_{\bar{D},C}$ from simulated *Dave*'s state

16:          send $(C, \text{sig}_{\bar{D},C})$ to $\mathcal{F}_{\text{Chan}}$

17:      **end if**

18: On $(\text{FUND YOU}, c, Bob, Charlie, Alice)$ by $\mathcal{F}_{\text{Chan}}$:

19:      simulate *Alice* receiving input $(\text{FUND YOU}, c, Bob)$ by *Charlie*

20:      ensure simulated *Alice* outputs (OK) to *Charlie*

21:      send (OK) to $\mathcal{F}_{\text{Chan}}$

22: On $(\text{FUND } c, \text{hops}, \texttt{sub\_parties} = (\text{fundee, counterparty}), \texttt{outer\_parties} = (Charlie, Dave), \texttt{funder} = Alice, \text{id})$ by $\mathcal{F}_{\text{Chan}}$:

23:      add the message data to `virtual_opening`

24:      simulate execution of line **??** of Fig. 6 with *Alice*// $\mathcal{S}$ knows *Bob* (*Alice*'s counterparty) through opening procedure

25:      send (OK) to $\mathcal{F}_{\text{Chan}}$

26: On $(\text{ALLOW FUND}, c, \texttt{sub\_parties}, \texttt{local\_funder} = L_i, \text{id}, i \stackrel{?}{=} |\text{hops}|)$ by $\mathcal{F}_{\text{Chan}}$'s *Alice* to *Charlie*:

27:      simulate receiving message with *Charlie* by *Alice* and all subsequent communication

28:      ensure the simulated *Charlie* sends (OK) to the simulated *Alice*

29:      intercept this message and send it to $\mathcal{F}_{\text{Chan}}$'s *Alice*

Fig. 21.

21

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

2:     retrieve and remove from `virtual_opening` the data marked with id

3:     simulate line 15 of Fig. 3 with *Alice* using this data

4:     ensure *Alice* completes execution of VChan() successfully

5:     send (OK) to $\mathcal{F}_{\text{Chan}}$

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

7:     retrieve and remove from `virtual_opening` the data marked with id

8:     simulate line 15 of Fig. 3 with *Alice* using this data

9:     ensure *Alice* completes execution of VChan() successfully

10:     extract from *Alice*'s state the new virtual funding TX V for pre-existing channel

11:     extract from *Alice*'s state the new commitment TX C that spends the on-chain funding TX

12:     send (V, C) to $\mathcal{F}_{\text{Chan}}$

13: On (FUND DONE, id) by $\mathcal{F}_{\text{Chan}}$'s *Alice* to *Charlie*:

14:     simulate receiving message with *Charlie* by *Alice* and all subsequent communication

15:     ensure the simulated *Charlie* sends (OK) to the simulated *Alice*

16:     intercept this message and send it to $\mathcal{F}_{\text{Chan}}$'s *Alice*

Fig. 22.

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

$$\forall G \in \mathcal{G} \text{ such that all parties in } G \text{ are honest,}$$

$$\sum_{P \in G} \text{logged-coins}(P) = \sum_{P \in G} \text{ledger-coins}(P) =$$

$$= \sum_{P \in G} \left(\text{top-up}(P) + \sum_{m \in \mathcal{T}} \text{pay-in}(m, P) - \sum_{m \in \mathcal{T}} \text{pay-out}(m, P) \right) ,$$

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

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

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

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

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

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

References