1 Anonymous Communication Functionality

Functionality $\mathcal{F}_{\text{anon}}^{\mathcal{D}}$ – pull model

[Orfeas: global or per-message delay?] State:

– `map`: a two-way mapping between physical and logical names
– `outbound`: a list of messages, recipients and remaining timeout
– t: the latest time, as reported by $\bar{\mathcal{G}}_{\text{CLOCK}}$

1: Upon receiving any message by *Alice* and prior to handling it:
2:     send (READ-CLOCK) to $\bar{\mathcal{G}}_{\text{CLOCK}}$ and assign reply to t'
3:     **if** $t' = t + 1$ **then**
4:         decrement the delay (third element) of each `outbound` entry if positive
5:         increment t
6:     **end if**

7: Upon receiving (REGISTER, `nym`) by *Alice*:
8:     ensure `logicalAddressOf`(*Alice*) $= \perp$ // **[Orfeas: remove?]**
9:     add (*Alice*, `nym`) to `map`
10:     send (REGISTER) as *Alice* to $\bar{\mathcal{G}}_{\text{CLOCK}}$

11: Upon receiving (SEND, `party`, M) from *Alice*:
12:     add (`physicalAddressOf`(`party`), M, `sample`($\mathcal{D}$)) to `outbound` // sender optional, can be part of payload

13: Upon receiving (FETCH) from *Alice*:
14:     add to `new` all $M : (Alice, M, 0) \in$ `outbound`, remove entries from `outbound`
15:     send (NEW-MESSAGES, `new`) to *Alice*

16: Forward (ADVANCE-CLOCK) messages to $\bar{\mathcal{G}}_{\text{CLOCK}}$

Fig. 1.

Functionality $\mathcal{F}_{\text{anon}}^{\text{push}}$

State:

− `map`: a two-way mapping between physical and logical names

1: Upon receiving (REGISTER, `nym`) by *Alice*:
2:      ensure `logicalAddressOf`(*Alice*) $= \bot$ // **[Orfeas: remove?]**
3:      add (*Alice*, `nym`) to `map`

4: Upon receiving (SEND, `party`, M) from *Alice*:
5:      send (PUSH, M) to `physicalAddressOf`(`party`) // sender optional, can be part of payload

---

Fig. 2.

2   Anonymous Communication Protocol

Functionality $\mathcal{F}_{\text{token}}$

Upon receiving (PASS-TOKEN, *Bob*) by *Alice*, send (PASS-TOKEN) to *Bob*

Fig. 3.

Protocol $\Pi_{\mathrm{anon}}^{\mathcal{F}_{\mathrm{token}}}$ – push model

Each player sends one CIRCULATE message to every player in $\mathcal{P}$ per payload. The first CIRCULATE message is sent to the next player in lexicographic order and the player continues in a round-robin fashion. TODO: write better description

1: Upon receiving (SEND, **receiver**, M) from $\mathcal{E}$:
2:     player $\xleftarrow{\$} \mathcal{P}$
3:     calculate the last player that will receive a CIRCULATE message and encrypt **receiver** for them, encrypt M for **receiver** and garbage for every other player
4:     **if** player $\neq$ us **then**
5:         send (PASS-TOKEN, player) to $\mathcal{F}_{\mathrm{token}}$
6:     **else**
7:         send (CIRCULATE, message of next player) to next player
8:     **end if**

9: Upon receiving (PASS-TOKEN) from $\mathcal{F}_{\mathrm{token}}$:
10:     encrypt garbage for every player
11:     send (CIRCULATE, message of next player) to next player

12: Upon receiving (CIRCULATE, C) from Bob:
13:     decrypt C to M
14:     **if** M is **receiver then**
15:         store **receiver** for sending (PASS-TOKEN, **receiver**) to $\mathcal{F}_{\mathrm{token}}$ upon receiving (CIRCULATE, C') for the last time for this particular payload
16:     **else if** M is a valid payload **then**
17:         store M for returning to $\mathcal{E}$ upon receiving (PASS-TOKEN) from $\mathcal{F}_{\mathrm{token}}$
18:     **end if**
19:     send (CIRCULATE, encrypted garbage) to next player

Fig. 4. Realises $\mathcal{F}_{\mathrm{anon}}$ in the global passive adversary case

3 Payment Network Functionality

Functionality $\mathcal{F}_{\text{PayNet}}$– interface

– from $\mathcal{E}$:
  - (REGISTER, delay, relayDelay)
  - (TOPPEDUP)
  - (OPENCHANNEL, *Alice*, *Bob*, *x*, *tid*)
  - (CHECKFORNEW, *Alice*, *Bob*, *tid*)
  - (PAY, *Bob*, *x*, $\overrightarrow{\text{path}}$, receipt)
  - (CLOSECHANNEL, receipt, *pchid*)
  - (FORCECLOSECHANNEL, receipt, *pchid*)
  - (POLL) - obsolete
  - (PUSHFULFILL, *pchid*) - obsolete
  - (PUSHADD, *pchid*) - obsolete
  - (COMMIT, *pchid*) - obsolete
  - (FULFILLONCHAIN) - obsolete
  - (GETNEWS)
– to $\mathcal{E}$:
  - (REGISTER, *Alice*, delay(*Alice*), relayDelay(*Alice*), pubKey)
  - (REGISTERED)
  - (NEWS, newChannels, closedChannels, updatesToReport)
– from $\mathcal{S}$:
  - (REGISTERDONE, *Alice*, pubKey)
  - (CHANNELANNOUNCED, *Alice*, $p_{Alice,F}$, $p_{Bob,F}$, *fchid*, *pchid*, *tid*)
  - (UPDATE, receipt, *Alice*) - obsolete
  - (CLOSEDCHANNEL, channel, *Alice*)
  - (RESOLVEPAYS, *payid*, charged) - obsolete
– to $\mathcal{S}$:
  - (REGISTER, *Alice*, delay, relayDelay)
  - (OPENCHANNEL, *Alice*, *Bob*, *x*, *fchid*, *tid*)
  - (CHANNELOPENED, *Alice*, *fchid*)
  - (PAY, *Alice*, *Bob*, *x*, $\overrightarrow{\text{path}}$, receipt, *payid*) - obsolete
  - (CONTINUE) - obsolete
  - (CLOSECHANNEL, *fchid*, *Alice*)
  - (FORCECLOSECHANNEL, *fchid*, *Alice*)
  - (POLL, Σ_{Alice}, *Alice*) - obsolete
  - (PUSHFULFILL, *pchid*, *Alice*) - obsolete
  - (PUSHADD, *pchid*, *Alice*) - obsolete
  - (COMMIT, *pchid*, *Alice*) - obsolete
  - (FULFILLONCHAIN, *t*, *Alice*) - obsolete

Fig. 5.

4

Functionality $\mathcal{F}_{\text{PayNet}}$ – registration and corruption

1: Initialisation:
2:     $\texttt{channels}, \texttt{pendingPay}, \texttt{pendingOpen}, \texttt{corrupted}, \Sigma \leftarrow \emptyset$

3: Upon receiving (REGISTER, delay, relayDelay) from *Alice*:
4:     $\texttt{delay}(Alice) \leftarrow$ delay // Must check chain at least once every $\texttt{delay}(Alice)$ blocks
5:     $\texttt{relayDelay}(Alice) \leftarrow$ relayDelay
6:     $\texttt{updatesToReport}(Alice), \texttt{newChannels}(Alice) \leftarrow \emptyset$
7:     $\texttt{polls}(Alice) \leftarrow \emptyset$
8:     $\texttt{focs}(Alice) \leftarrow \emptyset$
9:     send (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice*, store reply to Σ_{Alice}, add Σ_{Alice} to Σ and add largest block number to $\texttt{polls}(Alice)$
10:     $\texttt{checkClosed}(\Sigma_{Alice})$
11:     send (REGISTER, *Alice*, delay, relayDelay) to $\mathcal{S}$

12: Upon receiving (REGISTERDONE, *Alice*, pubKey) from $\mathcal{S}$:
13:     $\texttt{pubKey}(Alice) \leftarrow$ pubKey
14:     send (REGISTER, *Alice*, $\texttt{delay}(Alice)$, $\texttt{relayDelay}(Alice)$, pubKey) to *Alice*

15: Upon receiving (TOPPEDUP) from *Alice*:
16:     send (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and store reply to Σ_{Alice}
17:     $\texttt{checkClosed}(\Sigma_{Alice})$
18:     assign the sum of all output values that are exclusively spendable by *Alice* to $\texttt{onChainBalance}$
19:     send (REGISTERED) to *Alice*

20: Upon receiving any message (M) except for (REGISTER) or (TOPPEDUP) from *Alice*:
21:     **if** if haven't received (REGISTER) and (TOPPEDUP) from *Alice* (in this order) **then**
22:         send (INVALID, M) to *Alice* and ignore message
23:     **end if**

Fig. 6.

Functionality $\mathcal{F}_{\text{PayNet}}-$ open

1: Upon receiving (OPENCHANNEL, *Alice*, *Bob*, x, *tid*) from *Alice*:

2:     ensure *tid* hasn't been used by *Alice* for opening another channel before

3:     choose unique channel ID *fchid*

4:     $\texttt{pendingOpen}(\textit{fchid}) \leftarrow (\textit{Alice}, \textit{Bob}, x, \textit{tid})$

5:     send (OPENCHANNEL, *Alice*, *Bob*, x, *fchid*, *tid*) to $\mathcal{S}$

6: Upon receiving (CHANNELANNOUNCED, *Alice*, $p_{Alice,F}$, $p_{Bob,F}$, *fchid*, *pchid*, *tid*) from $\mathcal{S}$:

7:     ensure that there is a $\texttt{pendingOpen}(\textit{fchid})$ entry with temporary id *tid*

8:     add $p_{Alice,F}, p_{Bob,F}$, *pchid* and mark "*Alice* announced" to $\texttt{pendingOpen}(\textit{fchid})$

9: Upon receiving (CHECKFORNEW, *Alice*, *Bob*, *tid*) from *Alice*:

10:     ensure there is a matching $\texttt{channel}$ in $\texttt{pendingOpen}(\textit{fchid})$, marked with "*Alice* announced"

11:     $(\text{funder}, \text{fundee}, x, p_{Alice,F}, p_{Bob,F}) \leftarrow \texttt{pendingOpen}(\textit{fchid})$

12:     send (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and store reply to Σ_{Alice}

13:     $\texttt{checkClosed}(\Sigma_{Alice})$

14:     ensure that there is a TX $F \in \Sigma_{Alice}$ with a $(x, (p_{\text{funder},F} \wedge p_{\text{fundee},F}))$ output

15:     mark $\texttt{channel}$ with "waiting for FUNDINGLOCKED"

16:     send (FUNDINGLOCKED, *Alice*, Σ_{Alice}, *fchid*) to $\mathcal{S}$

17: Upon receiving (FUNDINGLOCKED, *fchid*) from $\mathcal{S}$:

18:     ensure a $\texttt{channel}$ is in $\texttt{pendingOpen}(\textit{fchid})$, marked with "waiting for FUNDINGLOCKED" and replace mark with "waiting for CHANNELOPENED"

19:     send (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Bob* and store reply to Σ_{Bob}

20:     $\texttt{checkClosed}(\Sigma_{Bob})$

21:     ensure that there is a TX $F \in \Sigma_{Bob}$ with a $(x, (p_{\text{funder},F} \wedge p_{\text{fundee},F}))$ output

22:     add $\texttt{receipt}(\texttt{channel})$ to $\texttt{newChannels}(\textit{Bob})$

23:     send (FUNDINGLOCKED, *Bob*, Σ_{Bob}, *fchid*) to $\mathcal{S}$

24: Upon receiving (CHANNELOPENED, *fchid*) from $\mathcal{S}$:

25:     ensure a $\texttt{channel}$ is in $\texttt{pendingOpen}(\textit{fchid})$, marked with "waiting for CHANNELOPENED" and remove mark

26:     $\text{offChainBalance}(\text{funder}) \leftarrow \text{offChainBalance}(\text{funder}) + x$

27:     $\text{onChainBalance}(\text{funder}) \leftarrow \text{onChainBalance}(\text{funder}) - x$

28:     $\texttt{channel} \leftarrow (\text{funder}, \text{fundee}, x, 0, 0, \textit{fchid}, \textit{pchid})$

29:     add $\texttt{channel}$ to $\texttt{channels}$

30:     add $\texttt{receipt}(\texttt{channel})$ to $\texttt{newChannels}(\textit{Alice})$

31:     clear $\texttt{pendingOpen}(\textit{fchid})$ entry

Fig. 7.

6

Functionality $\mathcal{F}_{\mathrm{PayNet}}-$ pay (updated)

1: Upon receiving $\left(\mathrm{PAY}, Bob, x, \overrightarrow{\mathbf{path}}\right)$ from *Alice*:

2:      ensure that $\overrightarrow{\mathbf{path}}$ consists of open channels that form a path of capacity at least x (in the right direction) from *Alice* to *Bob*

3:      starting on $|\overrightarrow{\mathbf{path}}|$ clock ticks after receiving this message, on every clock tick, channel $\in \overrightarrow{\mathbf{path}}$, reduce balance of party closer to payer by x and increase balance of party closer to payee by x in the next channel on the $\overrightarrow{\mathbf{path}}$ and add **receipt** of new balance to both parties' **updatesToReport**, starting from the unique channel in which the payee is participating

Fig. 8.

Functionality $\mathcal{F}_{\mathrm{PayNet}}-$ close

1: Upon receiving (CLOSECHANNEL, **receipt**, *pchid*) from *Alice*

2:      ensure that there is a **channel** $\in$ **channels** : **receipt** (**channel**) = **receipt** with ID *pchid*

3:      retrieve *fchid* from **channel**

4:      add (*fchid*, **receipt**(**channel**), ∞) to **pendingClose**(*Alice*)

5:      do not serve any other (PAY, CLOSECHANNEL) message from *Alice* for this channel

6:      send (CLOSECHANNEL, **receipt**, *pchid*, *Alice*) to $\mathcal{S}$

7: Upon receiving (FORCECLOSECHANNEL, **receipt**, *pchid*) from *Alice*

8:      retrieve *fchid* from **channel**

9:      add (*fchid*, **receipt**(**channel**), $\bot$) to **pendingClose**(*Alice*)

10:      do not serve any other (PAY, CLOSECHANNEL, FORCECLOSECHANNEL) message from *Alice* for this channel

11:      send (FORCECLOSECHANNEL, **receipt**, *pchid*, *Alice*) to $\mathcal{S}$

12: Upon receiving (CLOSEDCHANNEL, **channel**, *Alice*) from $\mathcal{S}$:

13:      remove any (*fchid* of channel, **receipt**(**channel**), ∞) from **pendingClose**(*Alice*)

14:      add (*fchid* of channel, **receipt**(**channel**), $\bot$) to **closedChannels**(*Alice*) // trust $\mathcal{S}$ here, check on **checkClosed**()

15:      send (CONTINUE) to $\mathcal{S}$

Fig. 9.

7

Functionality $\mathcal{F}_{\text{PayNet}}-$ `checkClosed()`

1: **function** checkClosed(Σ_{Alice}) // Called after every (READ), ensures requested closes eventually happen
2:   **if** there is any closing/commitment transaction in Σ_{Alice} with no corresponding entry in `pendingClose`($Alice$) $\cup$ `closedChannels`($Alice$) **then**
3:     add ($fchid$, `receipt`, $\perp$) to `closedChannels`($Alice$), where $fchid$ is the ID of the corresponding `channel`, `receipt` comes from the latest `channel` state
4:   **end if**
5:   **for all** entries ($fchid$, `receipt`, h) $\in$ `pendingClose`($Alice$) $\cup$ `closedChannels`($Alice$) **do**
6:     **if** there is a closing/commitment transaction in Σ_{Alice} for open `channel` with ID $fchid$ with a balance that corresponds to `receipt` **then**
7:       let x, y $Alice$'s and `channel` counterparty Bob's balances respectively
8:       offChainBalance ($Alice$) $\leftarrow$ offChainBalance ($Alice$) $- x$
9:       onChainBalance ($Alice$) $\leftarrow$ onChainBalance ($Alice$) $+ x$
10:       offChainBalance (Bob) $\leftarrow$ offChainBalance (Bob) $- y$
11:       onChainBalance (Bob) $\leftarrow$ onChainBalance (Bob) $+ y$
12:       remove `channel` from `channels` & entry from `pendingClose`($Alice$)
13:       **if** there is an ($fchid$, _, _) entry in `pendingClose`(Bob) **then**
14:         remove it from `pendingClose`(Bob)
15:       **end if**
16:     **else if** there is a tx in Σ_{Alice} that is not a closing/commitment tx and spends the funding tx of the `channel` with ID $fchid$ **then**
17:       halt // DS forgery
18:     **else if** there is a commitment transaction in block of height h in Σ_{Alice} for open `channel` with ID $fchid$ with a balance that does not correspond to the `receipt` and the delayed output has been spent by the counterparty **then**
19:       **if** `polls`($Alice$) contains an entry in $[h, h + \text{delay}(Alice) - 1]$ **then**
20:         halt
21:       **else**
22:         `negligent`($Alice$) $\leftarrow$ `true`
23:       **end if**
24:     **else if** there is no such closing/commitment transaction $\wedge$ $h = \perp$ **then**
25:       assign largest block number of Σ_{Alice} to h of entry
26:     **else if** there is no such closing/commitment transaction $\wedge$ $h \neq \perp$ $\wedge$ (largest block number of Σ_{Alice}) $\geq h + (2 + r)$ `windowSize` **then**
27:       halt
28:     **end if**
29:   **end for**
30:   **if** $Alice$ has no open channels in Σ_{Alice} AND `negligent`($Alice$) $=$ `false` **then**
31:     **if** offChainBalance($Alice$) $\neq 0$ OR onChainBalance($Alice$) is not equal to the total funds exclusively spendable by $Alice$ in Σ_{Alice} **then**
32:       halt
33:     **end if**
34:   **end if**
35: **end function**

Fig. 10.

8

```
┌─ Functionality ℱ_PayNet− get news (updated) ──────────────┐
│ 1: Upon receiving (GETNEWS) from Alice:                    │
│ 2:    clear newChannels(Alice), closedChannels(Alice),    │
│   updatesToReport(Alice)                                   │
│   and send them to Alice with message name NEWS, stripping│
│   fchid and h from closedChannels(Alice)                  │
└───────────────────────────────────────────────────────────┘
```

Fig. 11.

– The functionality above provides unobservability of off-chain payments and can be realised by a protocol in which all players communicate with every other player on every round – sending garbage to the ones with which they don't have to interact. Such a protocol has n^2 communication cost. Indistinguishability holds only in case of a global passive adversary (no corruptions).

– We can also assert unobservability for paths that consist of honest parties only in the case where there is a system-wide maximum path length l and corruptions activate only after $2l$ clock ticks. Both the functionality and the protocol would be the same.

– In case of a normal corruption model, the functionality has to leak the previous and next player on the path, along with the payment value, to a corrupted player that is to receive its message on this clock tick. Also, the functionality has to wait for confirmation from the corrupted player before sending the message to the next player (but this isn't strictly about privacy).

4   Lightning Protocol

Messages to and from *Bob* should be interpreted as SEND and RECEIVE messages to and from $\mathcal{F}_{\text{anon}}$ with *Bob* as the sender and the receiver respectively. *Bob* is a "logical" (LN-only) address, therefore *Alice* need not know *Bob*'s "real" address. This address is only known by $\mathcal{F}_{\text{anon}}$.

Protocol Π_{LN} (self is *Alice* always) – support

1: Initialisation:
2:     channels, pendingOpen, pendingPay, pendingClose ← ∅
3:     newChannels, closedChannels, updatesToReport ← ∅
4:     unclaimedOfferedHTLCs, unclaimedReceivedHTLCs, pendingGetPaid ← ∅

5: Upon receiving (REGISTER, delay, relayDelay) from $\mathcal{E}$:
6:     delay ← delay // Must check chain at least once every delay blocks
7:     relayDelay ← relayDelay
8:     send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to Σ_{Alice}
9:     $(pk_{Alice}, sk_{Alice}) \leftarrow$ KEYGEN ()
10:     send (REGISTER, *Alice*, delay, relayDelay, pk_{Alice}) to $\mathcal{E}$

11: Upon receiving (TOPPEDUP) from $\mathcal{E}$:
12:     send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to Σ_{Alice}
13:     assign the sum of all output values that are exclusively spendable by *Alice* to onChainBalance
14:     send (REGISTERED) to $\mathcal{E}$

15: Upon receiving any message (M) except for (REGISTER) or (TOPPEDUP):
16:     **if** if haven't received (REGISTER) and (TOPPEDUP) from $\mathcal{E}$ (in this order) **then**
17:         send (INVALID, M) to $\mathcal{E}$ and ignore message
18:     **end if**

19: **function** GetKeys
20:     $(p_F, s_F) \leftarrow$ KEYGEN () // For F output
21:     $(p_{\text{pay}}, s_{\text{pay}}) \leftarrow$ SETUP () // For com output to remote
22:     $(p_{\text{dpay}}, s_{\text{dpay}}) \leftarrow$ SETUP () // For com output to self
23:     $(p_{\text{htlc}}, s_{\text{htlc}}) \leftarrow$ SETUP () // For htlc output to self
24:     seed $\overset{\$}{\leftarrow} U(k)$ // For per com point
25:     $(p_{\text{rev}}, s_{\text{rev}}) \leftarrow$ MASTERKEYGEN () // For revocation in com
26:     **return** $((p_F, s_F), (p_{\text{pay}}, s_{\text{pay}}), (p_{\text{dpay}}, s_{\text{dpay}}),$
27:         $(p_{\text{htlc}}, s_{\text{htlc}}),$ seed, $(p_{\text{rev}}, s_{\text{rev}}))$
28: **end function**

Fig. 12.

Protocol Π_{LN} – OPENCHANNEL from $\mathcal{E}$

1: Upon receiving (OPENCHANNEL, $Alice$, Bob, x, tid) from $\mathcal{E}$:
2:      ensure tid hasn't been used for opening another channel before
3:      $((ph_F, sh_F), (phb_{\mathrm{pay}}, shb_{\mathrm{pay}}), (phb_{\mathrm{dpay}}, shb_{\mathrm{dpay}}),$
     $(phb_{\mathrm{htlc}}, shb_{\mathrm{htlc}}), \mathtt{seed}, (phb_{\mathrm{rev}}, shb_{\mathrm{rev}})) \leftarrow \mathtt{GetKeys}\,()$
4:      $\mathrm{prand}_1 \leftarrow \mathrm{PRF}\,(\mathtt{seed}, 1)$
5:      $(ph_{\mathrm{com},1}, sh_{\mathrm{com},1}) \leftarrow \mathrm{KEYSHAREGEN}\left(1^k; \mathrm{prand}_1\right)$
6:      associate keys with tid
7:      add $(Alice,\ Bob,\ x,\ \mathtt{tid},\ (ph_F, sh_F), (phb_{\mathrm{pay}}, shb_{\mathrm{pay}}), (phb_{\mathrm{dpay}}, shb_{\mathrm{dpay}})$
     $(phb_{\mathrm{htlc}}, shb_{\mathrm{htlc}}), (phb_{\mathrm{com},1}, shb_{\mathrm{com},1}), (phb_{\mathrm{rev}}, shb_{\mathrm{rev}}),\ tid)$ to $\mathtt{pendingOpen}$
8:      send (OPENCHANNEL,
     $x, \mathtt{delay} + (2 + r)\,\mathtt{windowSize}, ph_F, phb_{\mathrm{pay}}, phb_{\mathrm{dpay}}, phb_{\mathrm{htlc}}, ph_{\mathrm{com},1}, phb_{\mathrm{rev}}, tid)$
     to Bob

Fig. 13.

Protocol Π_{LN} – OPENCHANNEL from Bob

1: Upon receiving (OPENCHANNEL, x, $\mathtt{remoteDelay}$, pt_F, ptb_{pay}, ptb_{dpay}, ptb_{htlc},
     $pt_{\mathrm{com},1}$, ptb_{rev}, tid) from Bob:
2:      ensure tid has not been used yet with Bob
3:      $((ph_F, sh_F), (phb_{\mathrm{pay}}, shb_{\mathrm{pay}}), (phb_{\mathrm{dpay}}, shb_{\mathrm{dpay}}), (phb_{\mathrm{htlc}}, shb_{\mathrm{htlc}}), \mathtt{seed}$
     $(phb_{\mathrm{rev}}, shb_{\mathrm{rev}})) \leftarrow \mathtt{GetKeys}()$
4:      $\mathrm{prand}_1 \leftarrow \mathrm{PRF}\,(\mathtt{seed}, 1)$
5:      $(ph_{\mathrm{com},1}, sh_{\mathrm{com},1}) \leftarrow \mathrm{KEYSHAREGEN}\left(1^k; \mathrm{prand}_1\right)$
6:      associate keys with tid and store in $\mathtt{pendingOpen}$
7:      send (ACCEPTCHANNEL,
     $\mathtt{delay} + (2 + r)\,\mathtt{windowSize}, ph_F, phb_{\mathrm{pay}}, phb_{\mathrm{dpay}}, phb_{\mathrm{htlc}}, ph_{\mathrm{com},1}, phb_{\mathrm{rev}}, tid)$
     to Bob

Fig. 14.

Protocol Π_{LN} – ACCEPTCHANNEL

1: Upon receiving (ACCEPTCHANNEL, $\mathtt{remoteDelay}, pt_F, ptb_{\mathrm{pay}}, ptb_{\mathrm{dpay}}, ptb_{\mathrm{htlc}},$
   $pt_{\mathrm{com},1}, ptb_{\mathrm{rev}}, tid$) from Bob:
2:     ensure there is a temporary ID tid with Bob in $\mathtt{pendingOpen}$ on which
   ACCEPTCHANNEL hasn't been received
3:     associate received keys with tid
4:     send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}
5:     assign to $\mathtt{prevout}$ a transaction output found in Σ_{Alice} that is currently
   exclusively spendable by $Alice$ and has value $y \geq x$
6:     $F \leftarrow$ TX {input spends $\mathtt{prevout}$ with a SIGNDS(TX, sk_{Alice}), output 0
   pays $y - x$ to pk_{Alice}, output 1 pays x to $tid.ph_F \wedge pt_F$}
7:     $pchid \leftarrow \mathcal{H}(F)$
8:     add $pchid$ to $\mathtt{pendingOpen}$ entry with id tid
9:     $pt_{\mathrm{rev},1} \leftarrow$ COMBINEPUBKEY$(ptb_{\mathrm{rev}}, ph_{\mathrm{com},1})$
10:    $(ph_{\mathrm{dpay},1}, sh_{\mathrm{dpay},1}) \leftarrow$ KEYDER$(phb_{\mathrm{dpay}}, shb_{\mathrm{dpay}}, ph_{\mathrm{com},1})$
11:    $(ph_{\mathrm{pay},1}, sh_{\mathrm{pay},1}) \leftarrow$ KEYDER$(phb_{\mathrm{pay}}, shb_{\mathrm{pay}}, ph_{\mathrm{com},1})$
12:    $(ph_{\mathrm{htlc},1}, sh_{\mathrm{htlc},1}) \leftarrow$ KEYDER$(phb_{\mathrm{htlc}}, shb_{\mathrm{htlc}}, ph_{\mathrm{com},1})$
13:    $\mathtt{remoteCom} \leftarrow \mathtt{remoteCom}_1 \leftarrow$ TX {input: output 1 of F, outputs:
   $(x, ph_{\mathrm{pay},1}), (0, ph_{\mathrm{rev},1} \vee (pt_{\mathrm{dpay},1}, \mathtt{delay} + (2 + r) \mathtt{windowSize}$ relative))}
14:    $\mathtt{localCom} \leftarrow$ TX {input: output 1 of F, outputs:
   $(x, pt_{\mathrm{rev},1} \vee (ph_{\mathrm{dpay},1}, \mathtt{remoteDelay}$ relative)), $(0, pt_{\mathrm{pay},1})$}
15:    add $\mathtt{remoteCom}$ and $\mathtt{localCom}$ to channel entry in $\mathtt{pendingOpen}$
16:    sig $\leftarrow$ SIGNDS$(\mathtt{remoteCom}_1, sh_F)$
17:    $\mathtt{lastRemoteSigned} \leftarrow 0$
18:    send (FUNDINGCREATED, tid, $pchid$, sig) to Bob

Fig. 15.

12

Protocol Π_{LN} – FUNDINGCREATED

1: Upon receiving (FUNDINGCREATED, *tid*, *pchid*, $\mathtt{BobSig}_1$) from *Bob*:
2:     ensure there is a temporary ID *tid* with *Bob* in $\mathtt{pendingOpen}$ on which we have sent up to ACCEPTCHANNEL
3:     $ph_{\mathrm{rev},1} \leftarrow \text{COMBINEPUBKEY}(phb_{\mathrm{rev}}pt_{\mathrm{com},1})$
4:     $pt_{\mathrm{dpay},1} \leftarrow \text{PUBKEYDER}(ptb_{\mathrm{dpay}}, pt_{\mathrm{com},1})$
5:     $pt_{\mathrm{pay},1} \leftarrow \text{PUBKEYDER}(ptb_{\mathrm{pay}}, pt_{\mathrm{com},1})$
6:     $pt_{\mathrm{htlc},1} \leftarrow \text{PUBKEYDER}(ptb_{\mathrm{htlc}}, pt_{\mathrm{com},1})$
7:     $\mathtt{localCom} \leftarrow \mathtt{localCom}_1 \leftarrow \text{TX}$ {input: output 1 of F, outputs: $(x, pt_{\mathrm{pay},1})$, $(0, pt_{\mathrm{rev},1} \vee (ph_{\mathrm{dpay},1}, \mathtt{remoteDelay}$ relative)}
8:     ensure $\text{VERIFYDS}(\mathtt{BobSig}_1, \mathtt{localCom}_1, pt_F) = \mathtt{True}$
9:     $\mathtt{remoteCom} \leftarrow \mathtt{remoteCom}_1 \leftarrow \text{TX}$ {input: output 1 of F, outputs: $(x, ph_{\mathrm{rev},1} \vee (pt_{\mathrm{dpay},1}, \mathtt{delay} + (2+r)\,\mathtt{windowSize}$ relative)), $(0, ph_{\mathrm{pay},1})$}
10:     add $\mathtt{BobSig}_1, \mathtt{remoteCom}_1$ and $\mathtt{localCom}_1$ to channel entry in $\mathtt{pendingOpen}$
11:     $\mathrm{sig} \leftarrow \text{SIGNDS}(\mathtt{remoteCom}_1, sh_F)$
12:     mark channel as "broadcast, no FUNDINGLOCKED"
13:     $\mathtt{lastRemoteSigned}, \mathtt{lastLocalSigned} \leftarrow 0$
14:     send (FUNDINGSIGNED, *pchid*, sig) to *Bob*

Fig. 16.

Protocol Π_{LN} – FUNDINGSIGNED

1: Upon receiving (FUNDINGSIGNED, *pchid*, $\mathtt{BobSig}_1$) from *Bob*:
2:     ensure there is a channel ID *pchid* with *Bob* in $\mathtt{pendingOpen}$ on which we have sent up to FUNDINGCREATED
3:     ensure $\text{VERIFYDS}(\mathtt{BobSig}_1, \mathtt{localCom}, pb_F) = \mathtt{True}$
4:     $\mathtt{localCom}_1 \leftarrow \mathtt{localCom}$
5:     $\mathtt{lastLocalSigned} \leftarrow 0$
6:     add $\mathtt{BobSig}_1$ to channel entry in $\mathtt{pendingOpen}$
7:     $\mathrm{sig} \leftarrow \text{SIGNDS}(F, sk_{Alice})$
8:     mark *pchid* in $\mathtt{pendingOpen}$ as "broadcast, no FUNDINGLOCKED"
9:     send (SUBMIT, (sig, F)) to $\mathcal{G}_{\mathrm{Ledger}}$

Fig. 17.

Protocol Π_{LN} – CHECKFORNEW

1: Upon receiving (CHECKFORNEW, *Alice*, *Bob*, *tid*) from $\mathcal{E}$: // lnd polling daemon
2:    ensure there is a matching **channel** in **pendingOpen** with id *pchid*, with a "broadcast" and a "no FUNDINGLOCKED" mark, funded with x coins
3:    send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}
4:    ensure $\exists$ unspent TX in Σ_{Alice} with ID *pchid* and a $(x, ph_F \wedge pt_F)$ output
5:    $\mathrm{prand}_2 \leftarrow \mathrm{PRF}\,(\mathtt{seed}, 2)$
6:    $(ph_{\mathrm{com},2}, sh_{\mathrm{com},2}) \leftarrow \mathrm{KEYSHAREGEN}\,(1^k; \mathrm{prand}_2)$
7:    add TX to **channel** data
8:    replace "broadcast" mark in **channel** with "FUNDINGLOCKED sent"
9:    send (FUNDINGLOCKED, *pchid*, $ph_{\mathrm{com},2}$) to *Bob*

Fig. 18.

Protocol Π_{LN} – FUNDINGLOCKED

1: Upon receiving (FUNDINGLOCKED, *pchid*, $pt_{\mathrm{com},2}$) from *Bob*:
2:    ensure there is a **channel** with ID *pchid* with *Bob* in **pendingOpen** with a "no FUNDINGLOCKED" mark
3:    **if channel** is not marked with "FUNDINGLOCKED sent" **then** // i.e. marked with "broadcast"
4:        send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}
5:        ensure $\exists$ unspent TX in Σ_{Alice} with ID *pchid* and a $(x, ph_F \wedge pt_F)$ output
6:        add TX to **channel** data
7:        $\mathrm{prand}_2 \leftarrow \mathrm{PRF}\,(\mathtt{seed}, 2)$
8:        $(ph_{\mathrm{com},2}, sh_{\mathrm{com},2}) \leftarrow \mathrm{KEYSHAREGEN}\,(1^k; \mathrm{prand}_2)$
9:        generate 2nd remote delayed payment, htlc, payment keys
10:    **end if**
11:    replace "no FUNDINGLOCKED" mark in **channel** with "FUNDINGLOCKED received"
12:    move channel data from **pendingOpen** to **channels**
13:    add receipt of channel to **newChannels**, where
    receipt $\leftarrow (Alice : x, Bob : 0, pchid)$
14:    **if channel** is not marked with "FUNDINGLOCKED sent" **then**
15:        replace "broadcast" mark in **channel** with "FUNDINGLOCKED sent"
16:        send (FUNDINGLOCKED, *pchid*, $ph_{\mathrm{com},2}$) to *Bob*
17:    **end if**

Fig. 19.

14

Protocol Π_{LN} – poll

1: Upon receiving (POLL) from $\mathcal{E}$:

2: send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}

3: $\mathtt{toSubmit} \leftarrow \emptyset$

4: **for all** $\tau \in \mathtt{unclaimedOfferedHTLCs}$ **do**

5:    **if** input of τ has been spent **then** // by remote HTLC-success

6:      remove τ from $\mathtt{unclaimedOfferedHTLCs}$

7:      **if** we are intermediary **then**

8:        retrieve preimage R, $pchid'$ of previous channel on the path of the HTLC, and $\mathtt{HTLCNo}'$ of the corresponding HTLC$'$ in $pchid'$

9:        add $(\mathtt{HTLCNo}', R)$ to $\mathtt{pendingFulfills}_{pchid'}$

10:       **end if**

11:   **else if** input of τ has not been spent and timelock is over **then**

12:       remove τ from $\mathtt{unclaimedOfferedHTLCs}$

13:       add τ to $\mathtt{toSubmit}$

14:    **end if**

15: **end for**

16: run loop of Fig. **??**

17: **for all** honestly closed $\mathtt{remoteCom}_n$ that were processed above, with channel id $pchid$ **do**

18:    **for all** received HTLC outputs i of $\mathtt{remoteCom}_n$ **do**

19:      **if** there is an entry in $\mathtt{pendingFulfills}_{pchid}$ with the same $\mathtt{HTLCNo}$ and R **then**

20:        TX $\leftarrow$ {input: i HTLC output of $\mathtt{remoteCom}_n$ with $(ph_{\mathrm{htlc},n}, R)$ as method, output: pk_{Alice}}

21:        sig $\leftarrow$ SignIBS $(TX, sh_{\mathrm{htlc},n})$

22:        add (sig, TX) to $\mathtt{toSubmit}$

23:        remove entry from $\mathtt{pendingFulfills}_{pchid}$

24:      **end if**

25:    **end for**

26: **end for**

27: send (SUBMIT, $\mathtt{toSubmit}$) to $\mathcal{G}_{\mathrm{Ledger}}$

28: Upon receiving (GETNEWS) from $\mathcal{E}$:

29: clear $\mathtt{newChannels}$, $\mathtt{closedChannels}$, $\mathtt{updatesToReport}$ and send them to $\mathcal{E}$ with message name NEWS

Fig. 20.

Loop over closed channels for poll

1: **for all** remoteCom$_n$ ∈ Σ_{Alice} that spend F of a `channel` ∈ `channels` **do**
2:      **if** we do not have $sh_{\mathrm{rev},n}$ **then** // Honest closure
3:          **for all** unspent offered HTLC outputs i of remoteCom$_n$ **do**
4:              TX ← {input: i HTLC output of remoteCom$_n$ with $ph_{\mathrm{htlc},n}$ as method, output: pk_{Alice}}
5:              sig ← SIGNIBS $(TX, sh_{\mathrm{htlc},n})$
6:              **if** timelock has not expired **then**
7:                  add (sig, TX) to `unclaimedOfferedHTLCs`
8:              **else if** timelock has expired **then**
9:                  add (sig, TX) to `toSubmit`
10:             **end if**
11:         **end for**
12:         **for all** spent offered HTLC output i of remoteCom$_n$ **do**
13:             **if** we are intermediary **then**
14:                 retrieve preimage R, $pchid'$ of previous channel on the path of the HTLC, and `HTLCNo'` of the corresponding HTLC' in $pchid'$
15:                 add (`HTLCNo'`, R) to `pendingFulfills`$_{pchid'}$
16:             **end if**
17:         **end for**
18:     **else** // malicious closure
19:         rev ← TX {inputs: all remoteCom$_n$ outputs, choosing $ph_{\mathrm{rev},n}$ method, output: pk_{Alice}}
20:         sig ← SIGNCS (rev, $sh_{\mathrm{rev},n}$)
21:         add (sig, rev) to `toSubmit`
22:     **end if**
23:     add `receipt`(`channel`) to `closedChannels`
24:     remove `channel` from `channels`
25: **end for**

Fig. 21.

Protocol $\Pi_{\mathrm{LN}} - \mathrm{pay}$

1: Upon receiving (PAY, *Bob*, x, $\overrightarrow{\texttt{path}}$) from $\mathcal{E}$:
2:     ensure that $\overrightarrow{\texttt{path}}$ consists of syntactically valid (*pchid*, CltvExpiryDelta) pair // Payment completes only if
   $\forall$ honest $i \in \overrightarrow{\texttt{path}}, \texttt{CltvExpiryDelta}_i \geq 3k + \texttt{RelayDelay}_i$
3:     ensure that the first *pchid* $\in \overrightarrow{\texttt{path}}$ corresponds to an open
   channel $\in$ channels in which we own at least x in the irrevocably committed state.
4:     choose unique payment ID *payid* // unique for *Alice* and *Bob*
5:     add (*Bob*, x, $\overrightarrow{\texttt{path}}$, *payid*, "waiting for invoice") to pendingPay
6:     send (SENDINVOICE, *payid*) to *Bob*

7: Upon receiving (SENDINVOICE, *payid*) from *Bob*:
8:     ensure there is no (*Bob*, *payid*) entry in pendingGetPaid
9:     choose random, unique preimage R
10:    add (*Bob*, R, *payid*) to pendingGetPaid
11:    send (INVOICE, $\mathcal{H}(R)$, relayDelay $+ (2 + r)$ windowSize, *payid*) to *Bob*

Fig. 22.

> **Protocol** Π_{LN} – invoice

1: Upon receiving (INVOICE, h, minFinalCltvExpiry, *payid*) from *Bob*:
2:     ensure there is a (*Bob*, x, $\overrightarrow{\mathtt{path}}$, *payid*, "waiting for invoice") entry in pendingPay
3:     ensure h is valid (in the range of $\mathcal{H}$)
4:     retrieve CltvExpiryDeltas from $\overrightarrow{\mathtt{path}}$ and remove entry from pendingPay
5:     send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign largest block number to t
6:     $l \leftarrow |\left(\overrightarrow{\mathtt{path}}\right)|$
7:     $\mathtt{CltvExpiry}_l \leftarrow t + \mathtt{minFinalCltvExpiry}$
8:
    $\forall i \in \{1,\ldots,l-1\}, \mathtt{CltvExpiry}_{l-i} \leftarrow \mathtt{CltvExpiry}_{l-i+1} + \mathtt{CltvExpiryDelta}_{l-i+1}$

9:     ensure $\mathtt{CltvExpiry}_1 \geq \mathtt{CltvExpiry}_2 + \mathtt{relayDelay} + (2+r)\,\mathtt{windowSize}$
10:     $m \leftarrow$ the concatenation of $l\left(x, \mathtt{CltvExpiry}\right)$
11:     $(\mu_0, \delta_0) \leftarrow \mathtt{SphinxCreate}\left(m,\ \text{public keys of } \overrightarrow{\mathtt{path}} \text{ parties}\right)$
12:     let $\mathtt{remoteCom}_n$ the latest signed remote commitment tx with first $\overrightarrow{\mathtt{path}}$ member
13:     reduce simple payment output in remoteCom by x
14:     add an additional $(x,\ ph_{\mathrm{rev},n+1} \vee (ph_{\mathrm{htlc},n+1} \wedge pt_{\mathrm{htlc},n+1}$, on preimage of $h) \vee ph_{\mathrm{htlc},n+1}$, $\mathtt{CltvExpiry}_1$ absolute) output (all with $n+1$ keys) to remoteCom, marked with HTLCNo
15:     reduce delayed payment output in localCom by x
16:     add an additional $(x,\ pt_{\mathrm{rev},n+1} \vee (pt_{\mathrm{htlc},n+1}$, on preimage of $h) \vee (ph_{\mathrm{htlc},n+1} \wedge pt_{\mathrm{htlc},n+1}$, $\mathtt{CltvExpiry}_1$ absolute)) output (all with $n+1$ keys) to localCom, marked with HTLCNo
17:     increment $\mathtt{HTLCNo}_{pchid}$ by one and associate $x, h, pchid$ with it
18:     mark HTLCNo as "sender"
19:     send (UPDATEADDHTLC, first *pchid* of $\overrightarrow{\mathtt{path}}, \mathtt{HTLCNo}_{pchid}, x, h, \mathtt{CltvExpiry}_1, (\mu_0, \delta_0))$ to *pchid* channel counterparty

Fig. 23.

Protocol Π_{LN} – UPDATEADDHTLC

1: Upon receiving (UPDATEADDHTLC, *pchid*, HTLCNo, x, h, IncomingCltvExpiry, M) from *Bob*:

2:     run code of Fig. **??** – UPDATEADDHTLC checks

3:     increment HTLCNo$_{pchid}$ by one

4:     let remoteCom$_n$ the latest signed remote commitment tx

5:     reduce delayed payment output in remoteCom by x

6:     add an $(x, ph_{\mathrm{rev},n+1} \vee (ph_{\mathrm{htlc},n+1} \wedge pt_{\mathrm{htlc},n+1}$, IncomingCltvExpiry absolute$) \vee ph_{\mathrm{htlc},n+1}$, on preimage of h) htlc output (all with $n+1$ keys) to remoteCom, marked with HTLCNo

7:     reduce simple payment output in localCom by x

8:     add an $(x, pt_{\mathrm{rev},n+1} \vee pt_{\mathrm{htlc},n+1}$, IncomingCltvExpiry absolute$) \vee ((pt_{\mathrm{htlc},n+1} \wedge ph_{\mathrm{htlc},n+1}$, on preimage of h)) htlc output (all with $n+1$ keys) to remoteCom, marked with HTLCNo

9:     **if** $\delta = $ receiver **then**

10:         retrieve $R : \mathcal{H}(R) = h$ from pendingGetPaid and clear entry

11:         add (HTLCNo, R) to pendingFulfills$_{pchid}$

12:     **else if** $\delta \neq$ receiver **then** // Send HTLC to next hop

13:         retrieve *pchid*′ data

14:         let remoteCom$'_n$ the latest signed remote commitment tx

15:         reduce simple payment output in remoteCom′ by x

16:         add an additional $(x, ph_{\mathrm{rev},n+1} \vee (ph_{\mathrm{htlc},n+1} \wedge pt_{\mathrm{htlc},n+1}$, on preimage of h) $\vee ph_{\mathrm{htlc},n+1}$OutgoingCltvExpiry absolute) output (all with $n+1$ keys) to remoteCom′, marked with HTLCNo′

17:         reduce delayed payment output in localCom′ by x

18:         add an additional $(x, pt_{\mathrm{rev},n+1} \vee (pt_{\mathrm{htlc},n+1}$, on preimage of h) $\vee (pt_{\mathrm{htlc},n+1} \wedge ph_{\mathrm{htlc},n+1}$ OutgoingCltvExpiry absolute)) output (all with $n+1$ keys) to remoteCom′, marked with HTLCNo′

19:         increment HTLCNo′ by 1

20:         $M' \leftarrow$ SphinxPrepare (M, δ, sk_{Alice})

21:         add (HTLCNo′, x, h, OutgoingCltvExpiry, M') to pendingAdds$_{pchid'}$

22:     **end if**

Fig. 24.

Protocol Π_{LN} – UPDATEADDHTLC checks

1: ensure *pchid* corresponds to an open `channel` in `channels` where *Bob* has at least x
2: ensure $\mathtt{HTLCNo} = \mathtt{HTLCNo}_{pchid} + 1$
3: $(pchid', x', \mathtt{OutgoingCltvExpiry}, \delta) \leftarrow \mathtt{SphinxPeel}\,(sk_{Alice}, M)$
4: send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign largest block number to t
5: **if** $\delta = \mathtt{receiver}$ **then**
6:     ensure $pchid' = \perp, x = x', \mathtt{IncomingCltvExpiry} \geq \mathtt{OutgoingCltvExpiry} = \mathtt{minFinalCltvExpiry}$
7:     mark $\mathtt{HTLCNo}$ as "receiver"
8: **else** // We are an intermediary
9:     ensure $x = x', \mathtt{IncomingCltvExpiry} \geq \max\{\mathtt{OutgoingCltvExpiry}, t\} + \mathtt{relayDelay} + 2\,(2 + r)\,\mathtt{windowSize}$
10:     ensure $\mathtt{pchid}'$ corresponds to an open channel in `channels` where we have at least x
11:     mark $\mathtt{HTLCNo}$ as "intermediary"
12: **end if**

Fig. 25.

Protocol Π_{LN} – UPDATEFULFILLHTLC

1: Upon receiving (UPDATEFULFILLHTLC, *pchid*, HTLCNo, R) from *Bob*:
2:     **if** HTLCNo > lastRemoteSigned $\vee$ HTLCNo > lastLocalSigned $\vee$ $\mathcal{H}(R) \neq h$, where h is the hash in the HTLC with number HTLCNo **then**
3:         close channel (as in Fig. **??**)
4:         **return**
5:     **end if**
6:     ensure HTLCNo is an offered HTLC (localCom has h tied to a public key that we own)
7:     add value of HTLC to delayed payment of remoteCom
8:     remove HTLC output with number HTLCNo from remoteCom
9:     add value of HTLC to simple payment of localCom
10:     remove HTLC output with number HTLCNo from localCom
11:     **if** we have a channel *phcid'* that has a received HTLC with hash h with number HTLCNo' **then** // We are intermediary
12:         send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}
13:         **if** latest remoteCom$'_n \in \Sigma_{Alice}$ **then** // counterparty has gone on-chain
14:             TX $\leftarrow$ {input: (remoteCom' HTLC output with number HTLCNo', R), output: pk_{Alice}}
15:             sig $\leftarrow$ SIGNIBS (TX, $sh_{\mathrm{htlc},n}$)
16:             send (SUBMIT, (sig, TX)) to $\mathcal{G}_{\mathrm{Ledger}}$ // shouldn't be already spent by remote HTLCTimeout
17:         **else** // counterparty still off-chain
18:             // Not having the HTLC irrevocably committed is impossible (Fig. **??**, l. **??**)
19:             send (UPDATEFULFILLHTLC, *pchid'*, HTLCNo', R) to counterparty
20:         **end if**
21:     **end if**

Fig. 26.

Protocol Π_{LN} – COMMIT

1: Upon receiving (COMMIT, *pchid*) from $\mathcal{E}$:

2:      ensure that there is a `channel` $\in$ `channels` with ID *pchid*

3:      retrieve latest remote commitment tx `remoteCom`$_n$ in `channel`

4:      ensure `remoteCom` $\neq$ `remoteCom`$_n$ // there are uncommitted updates

5:      ensure `channel` is not marked as "waiting for REVOKEANDACK"

6:      send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign largest block number to t

7:      undo adding all outgoing HTLCs in `remoteCom` for which we are
intermediary and `IncomingCltvExpiry` $< t + $ `relayDelay` $+ (2 + r)$ `windowSize`

8:      `remoteCom`$_{n+1}$ $\leftarrow$ `remoteCom`

9:      ComSig $\leftarrow$ SIGNDS (`remoteCom`$_{n+1}$, sh_F)

10:     HTLCSigs $\leftarrow \emptyset$

11:     **for** i from `lastRemoteSigned` $+ 1$ to `HTLCNo` **do**

12:         `remoteHTLC`$_{n+1,i}$ $\leftarrow$ TX $\{$input: HTLC output i of `remoteCom`$_{n+1}$,
output: $(c_{\mathrm{htlc,i}}, ph_{\mathrm{rev},n+1} \vee (pt_{\mathrm{dpay},n+1},$ `delay` $+ (2 + r)$ `windowSize` relative$))\}$

13:         add SIGNIBS (`remoteHTLC`$_{n+1,i}$, $sh_{\mathrm{htlc},n+1}$) to HTLCSigs

14:     **end for**

15:     `lastRemoteSigned` $\leftarrow$ `HTLCNo`

16:     mark `channel` as "waiting for REVOKEANDACK"

17:     send (COMMITMENTSIGNED, *pchid*, ComSig, HTLCSigs) to *pchid*
counterparty

Fig. 27.

Protocol Π_{LN} – COMMITMENTSIGNED

1: Upon receiving (COMMITMENTSIGNED, *pchid*, comSig_{n+1}, HTLCSigs_{n+1}) from *Bob*:

2:     ensure that there is a `channel` $\in$ `channels` with ID *pchid* with *Bob*

3:     retrieve latest local commitment tx localCom_n in `channel`

4:     ensure $\text{localCom} \neq \text{localCom}_n$ and $\text{localCom} \neq \text{pendingLocalCom}$ // there are uncommitted updates

5:     **if** $\text{VERIFYDS}\left(\text{comSig}_{n+1}, \text{localCom}, pt_F\right) = \texttt{false} \lor |\text{HTLCSigs}_{n+1}| \neq$ $\text{HTLCNo} - \text{lastLocalSigned}$ **then**

6:         close channel (as in Fig. **??**)

7:         **return**

8:     **end if**

9:     **for** i from $\text{lastLocalSigned} + 1$ to HTLCNo **do**

10:         $\text{localHTLC}_{n+1,i} \leftarrow \text{TX}$ {input: HTLC output i of localCom, output: $(c_{\text{htlc,i}}, ph_{\text{rev},n+1} \lor (pt_{\text{dpay},n+1}, \texttt{remoteDelay}$ relative$))$}

11:         **if** $\text{VERIFYIBS}(\text{HTLCSigs}_{n+1,i}, \text{localHTLC}_{n+1,i}, pt_{\text{htlc},n+1}) = \texttt{false}$ **then**

12:             close channel (as in Fig. **??**)

13:             **return**

14:         **end if**

15:     **end for**

16:     $\text{lastLocalSigned} \leftarrow \text{HTLCNo}$

17:     $\text{pendingLocalCom} \leftarrow \text{localCom}$

18:     mark pendingLocalCom as "irrevocably committed"

19:     $\text{prand}_{n+2} \leftarrow \text{PRF}(\texttt{seed}, n+2)$

20:     $(ph_{\text{com},n+2}, sh_{\text{com},n+2}) \leftarrow \text{KEYSHAREGEN}\left(1^k; \text{prand}_{n+2}\right)$

21:     send (REVOKEANDACK, *pchid*, $\text{prand}_n, ph_{\text{com},n+2}$) to *Bob*

Fig. 28.

Protocol Π_{LN} – REVOKEANDACK

1: Upon receiving (REVOKEANDACK, *pchid*, $st_{\text{com},n}, pt_{\text{com},n+2}$) from *Bob*:

2:      ensure there is a `channel` $\in$ `channels` with *Bob* with ID *pchid* marked as "waiting for REVOKEANDACK"

3:      **if** TESTKEY $(pt_{\text{com},n}, st_{\text{com},n}) \neq 1$ **then** // wrong $st_{\text{com},n}$ - closing

4:          close channel (as in Fig. **??**)

5:          **return**

6:      **end if**

7:      mark `remoteCom`$_{n+1}$ as "irrevocably committed"

8:      `localCom`$_{n+1} \leftarrow$ `pendingLocalCom`

9:      unmark `channel`

10:     add `receipt(channel)` to `updatesToReport`

11:     $sh_{\text{rev},n} \leftarrow$ COMBINEKEY $(phb_{\text{rev}}, shb_{\text{rev}}, pt_{\text{com}n}, st_{\text{com},n})$

12:     $ph_{\text{rev},n+2} \leftarrow$ COMBINEPUBKEY $(phb_{\text{rev}}, pt_{\text{com},n+2})$

13:     $pt_{\text{rev},n+2} \leftarrow$ COMBINEPUBKEY $(ptb_{\text{rev}}, ph_{\text{com},n+2})$

14:     $(ph_{\text{dpay},n+2}, sh_{\text{dpay},n+2}) \leftarrow$ KEYDER $(phb_{\text{dpay}}, shb_{\text{dpay}}, ph_{\text{com},n+2})$

15:     $pt_{\text{dpay},n+2} \leftarrow$ PUBKEYDER $(ptb_{\text{dpay}}, pt_{\text{com},n+2})$

16:     $(ph_{\text{pay},n+2}, sh_{\text{pay},n+2}) \leftarrow$ KEYDER $(phb_{\text{pay}}, shb_{\text{pay}}, ph_{\text{com},n+2})$

17:     $pt_{\text{pay},n+2} \leftarrow$ PUBKEYDER $(ptb_{\text{pay}}, pt_{\text{com},n+2})$

18:     $(ph_{\text{htlc},n+2}, sh_{\text{htlc},n+2}) \leftarrow$ KEYDER $(phb_{\text{htlc}}, shb_{\text{htlc}}, ph_{\text{com},n+2})$

19:     $pt_{\text{htlc},n+2} \leftarrow$ PUBKEYDER $(ptb_{\text{htlc}}, pt_{\text{com},n+2})$

20:     **if** no outstanding HTLCs remain for this channel and the sequence for CLOSECHANNEL or SHUTDOWN (Fig. **??**) has been initiated **then**

21:         continue execution at Fig. **??**, l. **??** or l. **??** respectively

22:     **end if**

Fig. 29.

Protocol $\Pi_{\mathrm{LN}} - \mathrm{PUSH}$

1: Upon receiving (PUSHFULFILL, *pchid*) from $\mathcal{E}$:

2:     ensure that there is a `channel` $\in$ `channels` with ID *pchid*

3:     choose a member (`HTLCNo`, R) of $\mathbf{pendingFulfills}_{pchid}$ that is both in an "irrevocably committed" $\mathbf{remoteCom}_n$ and $\mathbf{localCom}_n$

4:     send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}

5:     remove (`HTLCNo`, R) from $\mathbf{pendingFulfills}_{pchid}$

6:     **if** $\mathbf{remoteCom}_n \notin \Sigma_{Alice}$ **then** // counterparty cooperative

7:         send (UPDATEFULFILLHTLC, *pchid*, `HTLCNo`, R) to *pchid* counterparty

8:     **else** // counterparty gone on-chain

9:         TX $\leftarrow$ {input: ($\mathbf{remoteCom}_n$ HTLC output with number `HTLCNo`, R), output: pk_{Alice}}

10:         sig $\leftarrow$ SIGNIBS (TX, $sh_{\mathrm{htlc},n}$)

11:         send (SUBMIT, (sig, TX)) to $\mathcal{G}_{\mathrm{Ledger}}$// shouldn't be already spent by remote `HTLCTimeout`

12:     **end if**

13: Upon receiving (PUSHADD, *pchid*) from $\mathcal{E}$:

14:     ensure that there is a `channel` $\in$ `channels` with ID *pchid*

15:     choose a member (`HTLCNo`, x, h, `CltvExpiry`, M) of $\mathbf{pendingAdds}_{pchid}$ that is both in an "irrevocably committed" $\mathbf{remoteCom}_n$ and $\mathbf{localCom}_n$

16:     remove chosen entry from $\mathbf{pendingAdds}_{pchid}$

17:     send (UPDATEADDHTLC, *pchid*, `HTLCNo`, x, h, `CltvExpiry`, M) to *pchid* counterparty

18: Upon receiving (FULFILLONCHAIN) from $\mathcal{E}$:

19:     send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign largest block number to t

20:     `toSubmit` $\leftarrow \emptyset$

21:     **for all** channels **do**

22:         **if** there exists an HTLC in latest $\mathbf{localCom}_n$ for which we have sent both UPDATEFULFILLHTLC and COMMITMENTSIGNED to a transaction without that HTLC to counterparty, but have not received the corresponding REVOKEANDACK AND the HTLC expires within $[t, t + (2 + r)$ `windowSize`] **then**

23:             add $\mathbf{localCom}_n$ of the channel and all corresponding valid `HTLC-success`es and `HTLC-timeout`s (for both $\mathbf{localCom}_n$ and $\mathbf{remoteCom}_n$[a]), along with their signatures to `toSubmit`

24:         **end if**

25:     **end for**

26:     send (SUBMIT, `toSubmit`) to $\mathcal{G}_{\mathrm{Ledger}}$

---
[a] Ensures funds retrieval if counterparty has gone on-chain

Fig. 30.

Protocol Π_{LN} – close unilaterally

1: Upon receiving (FORCECLOSECHANNEL, receipt, *pchid*) from $\mathcal{E}$:
2:     ensure receipt corresponds to an open channel $\in$ channels with ID *pchid*
3:     **if** the sequence for CLOSECHANNEL has been initiated and is pending on clearing all outstanding HTLCs **then**
4:         forget this "hook"
5:     **end if**
6:     assign latest channel sequence number to n
7:     HTLCs $\leftarrow \emptyset$
8:     **for** every HTLC output $\in$ localCom$_n$ with number i **do**
9:         sig $\leftarrow$ SIGNIBS $(\mathtt{localHTLC}_{n,i}, sh_{\mathrm{htlc},n})$
10:         add $(\mathrm{sig}, \mathtt{HTLCSigs}_{n,i}, \mathtt{localHTLC}_{n,i})$ to HTLCs
11:     **end for**
12:     sig $\leftarrow$ SIGNDS $(\mathtt{localCom}_n, sh_F)$
13:     add receipt(channel) to closedChannels
14:     remove channel from channels
15:     send $($SUBMIT, $(\mathrm{sig}, \mathtt{remoteSig}_n, \mathtt{localCom}_n), \mathrm{HTLCs})$ to $\mathcal{G}_{\mathrm{Ledger}}$

Fig. 31.

Protocol Π_{LN} – close cooperatively

1: Upon receiving (CLOSECHANNEL, receipt, *pchid*) from $\mathcal{E}$:

2:      ensure receipt corresponds to an open channel $\in$ channels with ID *pchid*

3:      stop serving any (PAY, CLOSECHANNEL) message from $\mathcal{E}$ for this channel.

4:      mark channel as "coop closing"

5:      **if** there are outstanding HTLC outputs in the latest $\texttt{localCom}_n$ **then** continue from here when there are none left

6:      **end if**

7:      send (SHUTDOWN, pk_{Alice}, *pchid*) to *Bob*

8: Upon receiving (SHUTDOWN, pk_{Bob}, *pchid*) from *Bob*:

9:      ensure there is an open channel $\in$ channels with *Bob* with ID *pchid*

10:      **if** channel is not marked "coop closing" **then**

11:        mark channel as "coop closing"

12:        **if** there are outstanding HTLC outputs in the latest $\texttt{localCom}_n$ **then** continue from here when there are none left

13:        **end if**

14:        send (SHUTDOWN, pk_{Alice}, *pchid*) to *Bob*

15:      **else**

16:        $Cl \leftarrow$ TX {input spends channel funding TX output, outputs pay x, y to pk_{Alice}, pk_{Bob} respectively, alphabetically ordered by some fixed encoding of the keys, where x is *Alice*'s and y is *Bob*'s balance in the latest channel state

17:        sig $\leftarrow$ SIGNDS(Cl, sk_{Alice})

18:        send (CLOSINGSIGNED, sig, *pchid*) to *Bob*

19:      **end if**

20: Upon receiving (CLOSINGSIGNED, bobSig, *pchid*) from *Bob*:

21:      ensure there is an open channel $\in$ channels with *Bob* with ID *pchid*

22:      ensure channel is marked as "coop closing"

23:      add receipt(channel) to closedChannels

24:      remove channel from channels

25:      $Cl \leftarrow$ TX {input spends channel funding TX output, outputs pay x, y to pk_{Alice}, pk_{Bob} respectively, alphabetically ordered by some fixed encoding of the keys, where x is *Alice*'s and y is *Bob*'s balance in the latest channel state

26:      ensure VERIFYDS(bobSig, Cl, pk_{Bob}) = True

27:      aliceSig $\leftarrow$ SIGNDS(Cl, sk_{Alice})

28:      sort aliceSig, bobSig according to the ordering of the respective keys to produce sig1, sig2

29:      send (SUBMIT, ((sig1, sig2), Cl)) to $\mathcal{G}_{\text{Ledger}}$

Fig. 32.