

1 Property Based approach to Payment Anonymity

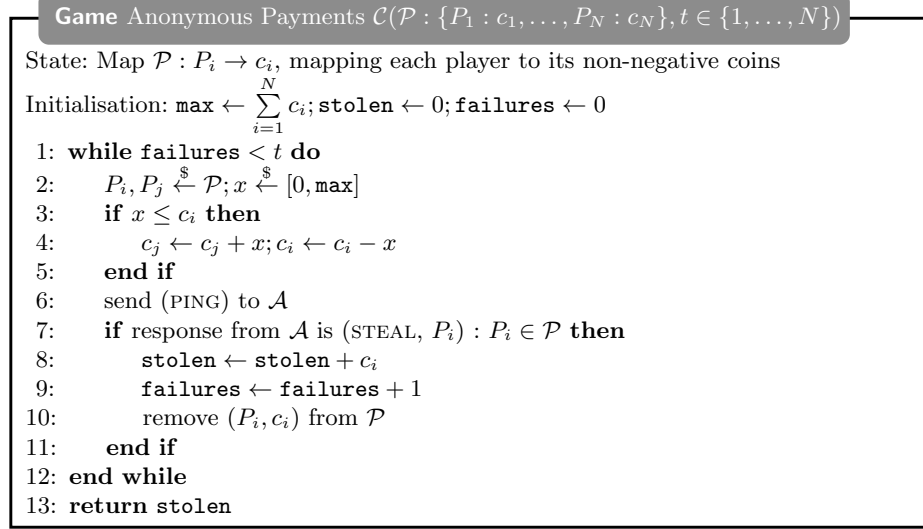


Fig. 1.

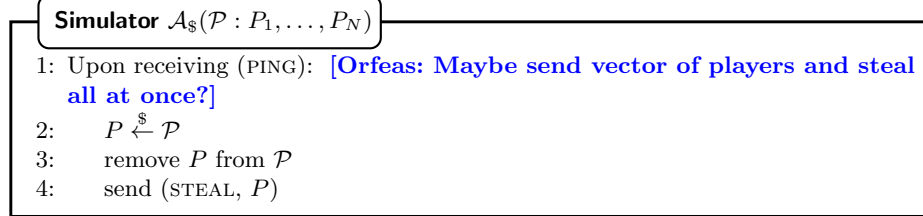


Fig. 2.

Definition 1 (Stealing Advantage).

$$Adv_{\text{AnonPay}}(\mathcal{C}; \mathcal{A}) = E(\mathcal{C}; \mathcal{A}) - E(\mathcal{C}; \mathcal{A}_{\$})$$

Definition 2 (Balance Privacy). \mathcal{C} has balance privacy if

$$\forall \text{ ITM } \mathcal{A}, Adv_{\text{AnonPay}}(\mathcal{C}; \mathcal{A}) < \text{negl}(\lambda) .$$

2 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 $\text{logicalAddressOf}(Alice) = \perp$ // [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 ($\text{physicalAddressOf}(\text{party})$, M , $\text{sample}(\mathcal{D})$) to **outbound** // sender optional, can be part of payload
 - 13: Upon receiving (FETCH) from *Alice*:
 - 14: add to **new** all $M : (Alice, M, 0) \in \text{outbound}$, remove entries from **outbound**
 - 15: send (NEW-MESSAGES, **new**) to *Alice*
 - 16: Forward (ADVANCE-CLOCK) messages to $\bar{\mathcal{G}}_{\text{clock}}$

Fig. 3.

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*) = \perp // [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. 4.

3 Anonymous Communication Protocol

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

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

Fig. 5.

Protocol $\Pi_{\text{anon}}^{\mathcal{F}_{\text{token}}}$ – 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 (REGISTER, **nym**) from \mathcal{E} :
- 2: /
- 3: Upon receiving (SEND, **receiver**, M) from \mathcal{E} :
- 4: **player** $\xleftarrow{\$} \mathcal{P}$
- 5: 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
- 6: **if player** \neq us **then**
- 7: send (PASS-TOKEN, **player**) to $\mathcal{F}_{\text{token}}$
- 8: **else**
- 9: send (CIRCULATE, message of next player) to next player
- 10: **end if**
- 11: Upon receiving (PASS-TOKEN) from $\mathcal{F}_{\text{token}}$:
- 12: **if toEnv = True then**
- 13: retrieve stored M and send (PUSH, M) to \mathcal{E}
- 14: **else**
- 15: encrypt garbage for every player
- 16: send (CIRCULATE, message of next player) to next player
- 17: **end if**
- 18: Upon receiving (CIRCULATE, C) from *Bob*:
- 19: decrypt C to M
- 20: **if M is receiver then**
- 21: store **receiver** for sending (PASS-TOKEN, **receiver**) to $\mathcal{F}_{\text{token}}$ upon receiving (CIRCULATE, C') for the last time for this particular payload
- 22: **else if M is a valid payload then**
- 23: store M
- 24: toEnv \leftarrow True
- 25: **end if**
- 26: send (CIRCULATE, encrypted garbage) to next player

Fig. 6. Realises $\mathcal{F}_{\text{anon}}$ in the global passive adversary case

4 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*, $p_{\text{Alice},F}$, $p_{\text{Bob},F}$, *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. 7.

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

- 1: Initialisation:
- 2: **channels**, **pendingPay**, **pendingOpen**, **corrupted**, $\Sigma \leftarrow \emptyset$
- 3: Upon receiving (REGISTER, delay, relayDelay) from *Alice*:
- 4: **delay**(*Alice*) \leftarrow delay // Must check chain at least once every
 delay(*Alice*) blocks
- 5: **relayDelay**(*Alice*) \leftarrow relayDelay
- 6: **updatesToReport**(*Alice*), **newChannels**(*Alice*) $\leftarrow \emptyset$
- 7: **polls**(*Alice*) $\leftarrow \emptyset$
- 8: **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 **polls**(*Alice*)
- 10: **checkClosed**(Σ_{Alice})
- 11: send (REGISTER, *Alice*, delay, relayDelay) to \mathcal{S}
- 12: Upon receiving (REGISTERDONE, *Alice*, pubKey) from \mathcal{S} :
- 13: **pubKey**(*Alice*) \leftarrow pubKey
- 14: send (REGISTER, *Alice*, **delay**(*Alice*), **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: **checkClosed**(Σ_{Alice})
- 18: assign the sum of all output values that are exclusively spendable by *Alice*
 to **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. 8.

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

- 6: Upon receiving (CHANNELANNOUNCED, *Alice*, $p_{\text{Alice},F}$, $p_{\text{Bob},F}$, *fchid*, *pchid*, *tid*) from \mathcal{S} :
- 7: ensure that there is a **pendingOpen**(*fchid*) entry with temporary id *tid*
- 8: add $p_{\text{Alice},F}$, $p_{\text{Bob},F}$, *pchid* and mark "*Alice* announced" to **pendingOpen**(*fchid*)

- 9: Upon receiving (CHECKFORNEW, *Alice*, *Bob*, *tid*) from *Alice*:
- 10: ensure there is a matching **channel** in **pendingOpen**(*fchid*), marked with "*Alice* announced"
- 11: (*funder*, *fundee*, *x*, $p_{\text{Alice},F}$, $p_{\text{Bob},F}$) \leftarrow **pendingOpen**(*fchid*)
- 12: send (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and store reply to Σ_{Alice}
- 13: **checkClosed**(Σ_{Alice})
- 14: ensure that there is a TX $F \in \Sigma_{\text{Alice}}$ with a $(x, (p_{\text{funder},F} \wedge p_{\text{fundee},F}))$ output
- 15: mark **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 **channel** is in **pendingOpen**(*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: **checkClosed**(Σ_{Bob})
- 21: ensure that there is a TX $F \in \Sigma_{\text{Bob}}$ with a $(x, (p_{\text{funder},F} \wedge p_{\text{fundee},F}))$ output
- 22: add **receipt**(**channel**) to **newChannels**(*Bob*)
- 23: send (FUNDINGLOCKED, *Bob*, Σ_{Bob} , *fchid*) to \mathcal{S}

- 24: Upon receiving (CHANNELOPENED, *fchid*) from \mathcal{S} :
- 25: ensure a **channel** is in **pendingOpen**(*fchid*), marked with "waiting for CHANNELOPENED" and remove mark
- 26: offChainBalance(*funder*) \leftarrow offChainBalance(*funder*) + *x*
- 27: onChainBalance(*funder*) \leftarrow onChainBalance(*funder*) - *x*
- 28: **channel** \leftarrow (*funder*, *fundee*, *x*, 0, 0, *fchid*, *pchid*)
- 29: add **channel** to **channels**
- 30: add **receipt**(**channel**) to **newChannels**(*Alice*)
- 31: clear **pendingOpen**(*fchid*) entry

Fig. 9.

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

- 1: Upon receiving $(\text{PAY}, \text{Bob}, x, \overrightarrow{\text{path}})$ from *Alice*:
- 2: ensure that $\overrightarrow{\text{path}}$ consists of open channels that form a path of capacity at least x (in the right direction) from *Alice* to *Bob*
- 3: starting on $|\overrightarrow{\text{path}}|$ clock ticks after receiving this message, on every clock tick, channel $\in \overrightarrow{\text{path}}$, 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 $\overrightarrow{\text{path}}$ and add **receipt** of new balance to both parties' **updatesToReport**, starting from the unique channel in which the payee is participating

Fig. 10.

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

- 1: Upon receiving $(\text{CLOSECHANNEL}, \text{receipt}, \text{pchid})$ from *Alice*
- 2: ensure that there is a $\text{channel} \in \text{channels} : \text{receipt}(\text{channel}) = \text{receipt}$ with ID pchid
- 3: retrieve fchid from channel
- 4: add $(\text{fchid}, \text{receipt}(\text{channel}), \infty)$ to $\text{pendingClose}(\text{Alice})$
- 5: do not serve any other $(\text{PAY}, \text{CLOSECHANNEL})$ message from *Alice* for this channel
- 6: send $(\text{CLOSECHANNEL}, \text{receipt}, \text{pchid}, \text{Alice})$ to \mathcal{S}
- 7: Upon receiving $(\text{FORCECLOSECHANNEL}, \text{receipt}, \text{pchid})$ from *Alice*
- 8: retrieve fchid from channel
- 9: add $(\text{fchid}, \text{receipt}(\text{channel}), \perp)$ to $\text{pendingClose}(\text{Alice})$
- 10: do not serve any other $(\text{PAY}, \text{CLOSECHANNEL}, \text{FORCECLOSECHANNEL})$ message from *Alice* for this channel
- 11: send $(\text{FORCECLOSECHANNEL}, \text{receipt}, \text{pchid}, \text{Alice})$ to \mathcal{S}
- 12: Upon receiving $(\text{CLOSEDCHANNEL}, \text{channel}, \text{Alice})$ from \mathcal{S} :
- 13: remove any $(\text{fchid}$ of channel, $\text{receipt}(\text{channel}), \infty)$ from $\text{pendingClose}(\text{Alice})$
- 14: add $(\text{fchid}$ of channel, $\text{receipt}(\text{channel}), \perp)$ to $\text{closedChannels}(\text{Alice})$ // trust \mathcal{S} here, check on $\text{checkClosed}()$
- 15: send (CONTINUE) to \mathcal{S}

Fig. 11.

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

```

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

```

Fig. 12.

Functionality $\mathcal{F}_{\text{PayNet-get news}}$ (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 *fhid* and *h* from
 closedChannels(*Alice*)

Fig. 13.

- 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).

5 Lightning Protocol

Messages to and from *Bob* should be interpreted as SEND and PUSH messages to and from $\mathcal{F}_{\text{anon}}^{\text{push}}$ 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 $\mathcal{F}_{\text{anon}}^{\text{push}}$.

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

```

1: Initialisation:
2:   channels, pendingOpen, pendingPay, pendingClose  $\leftarrow \emptyset$ 
3:   newChannels, closedChannels, updatesToReport  $\leftarrow \emptyset$ 
4:   unclaimedOfferedHTLCs, unclaimedReceivedHTLCs, pendingGetPaid  $\leftarrow \emptyset$ 

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

11: Upon receiving (TOPPEDUP) from  $\mathcal{E}$ :
12:  send (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign reply to  $\Sigma_{\text{Alice}}$ 
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:   $(p_F, s_F) \leftarrow \text{KEYGEN}()$  // For  $F$  output
21:   $(p_{\text{pay}}, s_{\text{pay}}) \leftarrow \text{SETUP}()$  // For com output to remote
22:   $(p_{\text{dpay}}, s_{\text{dpay}}) \leftarrow \text{SETUP}()$  // For com output to self
23:   $(p_{\text{htlc}}, s_{\text{htlc}}) \leftarrow \text{SETUP}()$  // For htlc output to self
24:  seed  $\xleftarrow{\$} U(k)$  // For per com point
25:   $(p_{\text{rev}}, s_{\text{rev}}) \leftarrow \text{MASTERKEYGEN}()$  // For revocation in com
26:  return  $((p_F, s_F), (p_{\text{pay}}, s_{\text{pay}}), (p_{\text{dpay}}, s_{\text{dpay}}),$ 
27:     $(p_{\text{htlc}}, s_{\text{htlc}}), \text{seed}, (p_{\text{rev}}, s_{\text{rev}}))$ 
28: end function

```

Fig. 14.

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: $((ph_F, sh_F), (ph_{b_{pay}}, sh_{b_{pay}}), (ph_{b_{dpay}}, sh_{b_{dpay}}), (ph_{b_{htlc}}, sh_{b_{htlc}}), \mathbf{seed}, (ph_{b_{rev}}, sh_{b_{rev}})) \leftarrow \mathbf{GetKeys}()$
- 4: $\mathbf{prand}_1 \leftarrow \mathbf{PRF}(\mathbf{seed}, 1)$
- 5: $(ph_{com,1}, sh_{com,1}) \leftarrow \mathbf{KEYSHAREGEN}(1^k; \mathbf{prand}_1)$
- 6: associate keys with *tid*
- 7: add (*Alice*, *Bob*, *x*, **tid**, (ph_F, sh_F) , $(ph_{b_{pay}}, sh_{b_{pay}})$, $(ph_{b_{dpay}}, sh_{b_{dpay}})$, $(ph_{b_{htlc}}, sh_{b_{htlc}})$, $(ph_{b_{com,1}}, sh_{b_{com,1}})$, $(ph_{b_{rev}}, sh_{b_{rev}})$, *tid*) to **pendingOpen**
- 8: send (OPENCHANNEL, *x*, **delay** + $(2 + r)$ **windowSize**, $ph_F, ph_{b_{pay}}, ph_{b_{dpay}}, ph_{b_{htlc}}, ph_{com,1}, ph_{b_{rev}}, tid$) to *Bob*

Fig. 15.

Protocol Π_{LN} – OPENCHANNEL from *Bob*

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

Fig. 16.

Protocol Π_{LN} – ACCEPTCHANNEL

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

Fig. 17.

Protocol Π_{LN} – FUNDINGCREATED

- 1: Upon receiving (FUNDINGCREATED, tid , $pchid$, $BobSig_1$) from *Bob*:
- 2: ensure there is a temporary ID tid with *Bob* in **pendingOpen** on which we have sent up to ACCEPTCHANNEL
- 3: $ph_{rev,1} \leftarrow \text{COMBINEPUBKEY}(ph_{rev}, pt_{com,1})$
- 4: $pt_{dpay,1} \leftarrow \text{PUBKEYDER}(pt_{dpay}, pt_{com,1})$
- 5: $pt_{pay,1} \leftarrow \text{PUBKEYDER}(pt_{pay}, pt_{com,1})$
- 6: $pt_{htlc,1} \leftarrow \text{PUBKEYDER}(pt_{htlc}, pt_{com,1})$
- 7: $localCom \leftarrow localCom_1 \leftarrow \text{TX}$ {input: output 1 of F , outputs: $(x, pt_{pay,1})$, $(0, pt_{rev,1} \vee (ph_{dpay,1}, \text{remoteDelay relative}))$ }
- 8: ensure $\text{VERIFYDS}(BobSig_1, localCom_1, pt_F) = \text{True}$
- 9: $remoteCom \leftarrow remoteCom_1 \leftarrow \text{TX}$ {input: output 1 of F , outputs: $(x, ph_{rev,1} \vee (pt_{dpay,1}, \text{delay} + (2 + r)\text{windowSize relative}))$, $(0, ph_{pay,1})$ }
- 10: add $BobSig_1$, $remoteCom_1$ and $localCom_1$ to channel entry in **pendingOpen**
- 11: $sig \leftarrow \text{SIGNDS}(remoteCom_1, sh_F)$
- 12: mark channel as “broadcast, no FUNDINGLOCKED”
- 13: $lastRemoteSigned, lastLocalSigned \leftarrow 0$
- 14: send (FUNDINGSIGNED, $pchid$, sig) to *Bob*

Fig. 18.

Protocol Π_{LN} – FUNDINGSIGNED

- 1: Upon receiving (FUNDINGSIGNED, $pchid$, $BobSig_1$) from *Bob*:
- 2: ensure there is a channel ID $pchid$ with *Bob* in **pendingOpen** on which we have sent up to FUNDINGCREATED
- 3: ensure $\text{VERIFYDS}(BobSig_1, localCom, pb_F) = \text{True}$
- 4: $localCom_1 \leftarrow localCom$
- 5: $lastLocalSigned \leftarrow 0$
- 6: add $BobSig_1$ to channel entry in **pendingOpen**
- 7: $sig \leftarrow \text{SIGNDS}(F, sk_{Alice})$
- 8: mark $pchid$ in **pendingOpen** as “broadcast, no FUNDINGLOCKED”
- 9: send (SUBMIT, (sig, F)) to \mathcal{G}_{Ledger}

Fig. 19.

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}_{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: $\text{prand}_2 \leftarrow \text{PRF}(\text{seed}, 2)$
- 6: $(ph_{com,2}, sh_{com,2}) \leftarrow \text{KEYSHAREGEN}(1^k; \text{prand}_2)$
- 7: add TX to **channel** data
- 8: replace “broadcast” mark in **channel** with “FUNDINGLOCKED sent”
- 9: send (FUNDINGLOCKED, *pchid*, $ph_{com,2}$) to *Bob*

Fig. 20.

Protocol Π_{LN} – FUNDINGLOCKED

- 1: Upon receiving (FUNDINGLOCKED, *pchid*, $pt_{com,2}$) 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}_{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: $\text{prand}_2 \leftarrow \text{PRF}(\text{seed}, 2)$
- 8: $(ph_{com,2}, sh_{com,2}) \leftarrow \text{KEYSHAREGEN}(1^k; \text{prand}_2)$
- 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
 $\text{receipt} \leftarrow (Alice : x, Bob : 0, pchid)$
- 14: **if** **channel** is not marked with “FUNDINGLOCKED sent” **then**
- 15: replace “broadcast” mark in **channel** with “FUNDINGLOCKED sent”
- 16: send (FUNDINGLOCKED, *pchid*, $ph_{com,2}$) to *Bob*
- 17: **end if**

Fig. 21.

Protocol $\Pi_{LN} - \text{poll}$

```

1: Upon receiving (POLL) from  $\mathcal{E}$ :
2:   send (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign reply to  $\Sigma_{\text{Alice}}$ 
3:    $\text{toSubmit} \leftarrow \emptyset$ 
4:   for all  $\tau \in \text{unclaimedOfferedHTLCs}$  do
5:     if input of  $\tau$  has been spent then // by remote HTLC-success
6:       remove  $\tau$  from  $\text{unclaimedOfferedHTLCs}$ 
7:       if we are intermediary then
8:         retrieve preimage  $R$ ,  $\text{pchid}'$  of previous channel on the path of
           the HTLC, and  $\text{HTLCNo}'$  of the corresponding HTLC' in  $\text{pchid}'$ 
9:         add  $(\text{HTLCNo}', R)$  to  $\text{pendingFulfills}_{\text{pchid}'}$ 
10:      end if
11:    else if input of  $\tau$  has not been spent and timelock is over then
12:      remove  $\tau$  from  $\text{unclaimedOfferedHTLCs}$ 
13:      add  $\tau$  to  $\text{toSubmit}$ 
14:    end if
15:  end for
16:  run loop of Fig. ??
17:  for all honestly closed  $\text{remoteCom}_n$  that were processed above, with
    channel id  $\text{pchid}$  do
18:    for all received HTLC outputs  $i$  of  $\text{remoteCom}_n$  do
19:      if there is an entry in  $\text{pendingFulfills}_{\text{pchid}}$  with the same HTLCNo
        and  $R$  then
20:         $\text{TX} \leftarrow \{\text{input: } i \text{ HTLC output of } \text{remoteCom}_n \text{ with } (ph_{\text{htlc},n}, R)$ 
        as method, output:  $pk_{\text{Alice}}\}$ 
21:         $\text{sig} \leftarrow \text{SIGNIBS}(\text{TX}, sh_{\text{htlc},n})$ 
22:        add  $(\text{sig}, \text{TX})$  to  $\text{toSubmit}$ 
23:        remove entry from  $\text{pendingFulfills}_{\text{pchid}}$ 
24:      end if
25:    end for
26:  end for
27:  send (SUBMIT,  $\text{toSubmit}$ ) to  $\mathcal{G}_{\text{Ledger}}$ 

28: Upon receiving (GETNEWS) from  $\mathcal{E}$ :
29:  clear  $\text{newChannels}$ ,  $\text{closedChannels}$ ,  $\text{updatesToReport}$  and send them to  $\mathcal{E}$ 
    with message name NEWS

```

Fig. 22.

Loop over closed channels for poll

```

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

```

Fig. 23.

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

- 1: Upon receiving $(\text{PAY}, \text{Bob}, x, \overrightarrow{\text{path}})$ from \mathcal{E} :
- 2: ensure that $\overrightarrow{\text{path}}$ consists of syntactically valid $(\text{pchid}, \text{CltvExpiryDelta})$ pair // Payment completes only if
 $\forall \text{ honest } i \in \overrightarrow{\text{path}}, \text{CltvExpiryDelta}_i \geq 3k + \text{RelayDelay}_i$
- 3: ensure that the first $\text{pchid} \in \overrightarrow{\text{path}}$ corresponds to an open $\text{channel} \in \text{channels}$ 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 $(\text{Bob}, x, \overrightarrow{\text{path}}, \text{payid}, \text{"waiting for invoice"})$ to pendingPay
- 6: send $(\text{SENDINVOICE}, \text{payid})$ to *Bob*

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

Fig. 24.

Protocol Π_{LN} – invoice

- 1: Upon receiving (INVOICE, h , $\text{minFinalCltvExpiry}$, payid) from *Bob*:
- 2: ensure there is a (*Bob*, x , $\overrightarrow{\text{path}}$, payid , “waiting for invoice”) entry in **pendingPay**
- 3: ensure h is valid (in the range of \mathcal{H})
- 4: retrieve CltvExpiryDeltas from $\overrightarrow{\text{path}}$ and remove entry from **pendingPay**
- 5: send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign largest block number to t
- 6: $l \leftarrow |\overrightarrow{\text{path}}|$
- 7: $\text{CltvExpiry}_l \leftarrow t + \text{minFinalCltvExpiry}$
- 8: $\forall i \in \{1, \dots, l-1\}, \text{CltvExpiry}_{l-i} \leftarrow \text{CltvExpiry}_{l-i+1} + \text{CltvExpiryDelta}_{l-i+1}$
- 9: ensure $\text{CltvExpiry}_1 \geq \text{CltvExpiry}_2 + \text{relayDelay} + (2+r)\text{windowSize}$
- 10: $m \leftarrow$ the concatenation of $l(x, \text{CltvExpiry})$
- 11: $(\mu_0, \delta_0) \leftarrow \text{SphinxCreate}(m, \text{public keys of } \overrightarrow{\text{path}} \text{ parties})$
- 12: let **remoteCom** $_n$ the latest signed remote commitment tx with first $\overrightarrow{\text{path}}$ member
- 13: reduce simple payment output in **remoteCom** by x
- 14: add an additional $(x, ph_{\text{rev}, n+1} \vee (ph_{\text{htlc}, n+1} \wedge pt_{\text{htlc}, n+1}, \text{on preimage of } h) \vee ph_{\text{htlc}, n+1}, \text{CltvExpiry}_1 \text{ absolute})$ output (all with $n+1$ keys) to **remoteCom**, marked with HTLCNo
- 15: reduce delayed payment output in **localCom** by x
- 16: add an additional $(x, pt_{\text{rev}, n+1} \vee (pt_{\text{htlc}, n+1}, \text{on preimage of } h) \vee (ph_{\text{htlc}, n+1} \wedge pt_{\text{htlc}, n+1}, \text{CltvExpiry}_1 \text{ absolute}))$ output (all with $n+1$ keys) to **localCom**, marked with HTLCNo
- 17: increment $\text{HTLCNo}_{\text{pchid}}$ by one and associate x, h, pchid with it
- 18: mark HTLCNo as “sender”
- 19: send (UPDATEADDHTLC, first pchid of $\overrightarrow{\text{path}}, \text{HTLCNo}_{\text{pchid}}, x, h, \text{CltvExpiry}_1, (\mu_0, \delta_0)$) to pchid channel counterparty

Fig. 25.

Protocol $\Pi_{LN} - \text{UPDATEADDHTLC}$

- 1: Upon receiving $(\text{UPDATEADDHTLC}, pchid, \text{HTLCNo}, x, h, \text{IncomingCltvExpiry}, M)$ 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 $(x, ph_{\text{rev},n+1} \vee (ph_{\text{htlc},n+1} \wedge pt_{\text{htlc},n+1}, \text{IncomingCltvExpiry absolute}) \vee ph_{\text{htlc},n+1}, \text{ on preimage of } h)$ htlc output (all with $n + 1$ keys) to remoteCom , marked with HTLCNo
- 7: reduce simple payment output in localCom by x
- 8: add an $(x, pt_{\text{rev},n+1} \vee pt_{\text{htlc},n+1}, \text{IncomingCltvExpiry absolute}) \vee ((pt_{\text{htlc},n+1} \wedge ph_{\text{htlc},n+1}, \text{ on preimage of } h))$ htlc output (all with $n + 1$ keys) to remoteCom , marked with HTLCNo
- 9: **if** $\delta = \text{receiver}$ **then**
- 10: retrieve $R : \mathcal{H}(R) = h$ from pendingGetPaid and clear entry
- 11: add (HTLCNo, R) to $\text{pendingFulfills}_{pchid}$
- 12: **else if** $\delta \neq \text{receiver}$ **then** // Send HTLC to next hop
- 13: retrieve $pchid'$ data
- 14: let $\text{remoteCom}'_n$ the latest signed remote commitment tx
- 15: reduce simple payment output in $\text{remoteCom}'$ by x
- 16: add an additional $(x, ph_{\text{rev},n+1} \vee (ph_{\text{htlc},n+1} \wedge pt_{\text{htlc},n+1}, \text{ on preimage of } h) \vee ph_{\text{htlc},n+1} \text{OutgoingCltvExpiry absolute})$ output (all with $n + 1$ keys) to $\text{remoteCom}'$, marked with HTLCNo'
- 17: reduce delayed payment output in $\text{localCom}'$ by x
- 18: add an additional $(x, pt_{\text{rev},n+1} \vee (pt_{\text{htlc},n+1}, \text{ on preimage of } h) \vee (pt_{\text{htlc},n+1} \wedge ph_{\text{htlc},n+1} \text{OutgoingCltvExpiry absolute}))$ output (all with $n + 1$ keys) to $\text{remoteCom}'$, marked with HTLCNo'
- 19: increment HTLCNo' by 1
- 20: $M' \leftarrow \text{SphinxPrepare}(M, \delta, sk_{\text{Alice}})$
- 21: add $(\text{HTLCNo}', x, h, \text{OutgoingCltvExpiry}, M')$ to $\text{pendingAdds}_{pchid'}$
- 22: **end if**

Fig. 26.

Protocol Π_{LN} – UPDATEADDHTLC checks

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

Fig. 27.

Protocol $\Pi_{LN} - \text{UPDATEFULFILLHTLC}$

```

1: Upon receiving (UPDATEFULFILLHTLC,  $pcid$ , 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  $pcid'$  that has a received HTLC with hash  $h$  with
   number HTLCNo' then // We are intermediary
12:    send (READ) to  $\mathcal{G}_{\text{Ledger}}$  and assign reply to  $\Sigma_{\text{Alice}}$ 
13:    if latest remoteCom' $_n \in \Sigma_{\text{Alice}}$  then // counterparty has gone on-chain
14:      TX  $\leftarrow$  {input: (remoteCom' HTLC output with number HTLCNo',  $R$ ),
        output:  $pk_{\text{Alice}}$ }
15:      sig  $\leftarrow$  SIGNIBS (TX,  $sh_{\text{htlc},n}$ )
16:      send (SUBMIT, (sig, TX)) to  $\mathcal{G}_{\text{Ledger}}$  // 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,  $pcid'$ , HTLCNo',  $R$ ) to counterparty
20:    end if
21:  end if

```

Fig. 28.

Protocol $\Pi_{LN} - \text{COMMIT}$

- 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 $\mathcal{G}_{\text{Ledger}}$ and assign largest block number to t
- 7: undo adding all outgoing HTLCs in **remoteCom** for which we are
intermediary and $\text{IncomingCltvExpiry} < t + \text{relayDelay} + (2 + r) \text{windowSize}$
- 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: ($c_{\text{htlc},i}$, $ph_{\text{rev},n+1} \vee (pt_{\text{dpay},n+1}, \text{delay} + (2 + r) \text{windowSize relative})$)}
- 13: add **SIGNIBS**(**remoteHTLC_{n+1,i}**, $sh_{\text{htlc},n+1}$) 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. 29.

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  $localCom \neq localCom_n$  and  $localCom \neq pendingLocalCom$  // there
   are uncommitted updates
5:   if  $VERIFYDS(comSig_{n+1}, localCom, pt_F) = false \vee |HTLCSigs_{n+1}| \neq$ 
    $HTLCNo - lastLocalSigned$  then
6:     close channel (as in Fig. ??)
7:     return
8:   end if
9:   for  $i$  from  $lastLocalSigned + 1$  to  $HTLCNo$  do
10:     $localHTLC_{n+1,i} \leftarrow TX \{input: HTLC \text{ output } i \text{ of } localCom, output:$ 
    $(C_{htlc,i}, ph_{rev,n+1} \vee (pt_{dpay,n+1}, remoteDelay \text{ relative}))\}$ 
11:    if  $VERIFYIBS(HTLCSigs_{n+1,i}, localHTLC_{n+1,i}, pt_{htlc,n+1}) = false$  then
12:      close channel (as in Fig. ??)
13:      return
14:    end if
15:  end for
16:   $lastLocalSigned \leftarrow HTLCNo$ 
17:   $pendingLocalCom \leftarrow localCom$ 
18:  mark  $pendingLocalCom$  as “irrevocably committed”
19:   $prand_{n+2} \leftarrow PRF(seed, n + 2)$ 
20:   $(ph_{com,n+2}, sh_{com,n+2}) \leftarrow KEYSHAREGEN(1^k; prand_{n+2})$ 
21:  send (REVOKEANDACK,  $pchid$ ,  $prand_n$ ,  $ph_{com,n+2}$ ) to Bob

```

Fig. 30.

Protocol Π_{LN} – REVOKEANDACK

```

1: Upon receiving (REVOKEANDACK,  $pchid$ ,  $st_{com,n}$ ,  $pt_{com,n+2}$ ) from Bob:
2:   ensure there is a channel  $\in$  channels with Bob with ID  $pchid$  marked as
   “waiting for REVOKEANDACK”
3:   if TESTKEY( $pt_{com,n}$ ,  $st_{com,n}$ )  $\neq$  1 then // wrong  $st_{com,n}$  - 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:   $sh_{rev,n} \leftarrow$  COMBINEKEY( $phb_{rev}$ ,  $shb_{rev}$ ,  $pt_{com,n}$ ,  $st_{com,n}$ )
12:   $ph_{rev,n+2} \leftarrow$  COMBINEPUBKEY( $phb_{rev}$ ,  $pt_{com,n+2}$ )
13:   $pt_{rev,n+2} \leftarrow$  COMBINEPUBKEY( $ptb_{rev}$ ,  $ph_{com,n+2}$ )
14:  ( $ph_{dpay,n+2}$ ,  $sh_{dpay,n+2}$ )  $\leftarrow$  KEYDER( $phb_{dpay}$ ,  $shb_{dpay}$ ,  $ph_{com,n+2}$ )
15:   $pt_{dpay,n+2} \leftarrow$  PUBKEYDER( $ptb_{dpay}$ ,  $pt_{com,n+2}$ )
16:  ( $ph_{pay,n+2}$ ,  $sh_{pay,n+2}$ )  $\leftarrow$  KEYDER( $phb_{pay}$ ,  $shb_{pay}$ ,  $ph_{com,n+2}$ )
17:   $pt_{pay,n+2} \leftarrow$  PUBKEYDER( $ptb_{pay}$ ,  $pt_{com,n+2}$ )
18:  ( $ph_{htlc,n+2}$ ,  $sh_{htlc,n+2}$ )  $\leftarrow$  KEYDER( $phb_{htlc}$ ,  $shb_{htlc}$ ,  $ph_{com,n+2}$ )
19:   $pt_{htlc,n+2} \leftarrow$  PUBKEYDER( $ptb_{htlc}$ ,  $pt_{com,n+2}$ )
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. 31.

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

- 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 **pendingFulfills** $_{pchid}$ that is both in an “irrevocably committed” **remoteCom** $_n$ and **localCom** $_n$
- 4: send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to Σ_{Alice}
- 5: remove (HTLCNo, R) from **pendingFulfills** $_{pchid}$
- 6: **if** **remoteCom** $_n \notin \Sigma_{\text{Alice}}$ **then** // counterparty cooperative
- 7: send (UPDATEFULFILLHTLC, $pchid$, HTLCNo, R) to $pchid$ counterparty
- 8: **else** // counterparty gone on-chain
- 9: TX \leftarrow {input: (**remoteCom** $_n$ HTLC output with number HTLCNo, R),
output: pk_{Alice} }
- 10: sig \leftarrow SIGNIBS (TX, $sh_{\text{htlc},n}$)
- 11: send (SUBMIT, (sig, TX)) to $\mathcal{G}_{\text{Ledger}}$ // 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, $x, h, \text{CltvExpiry}, M$) of **pendingAdds** $_{pchid}$ that is
both in an “irrevocably committed” **remoteCom** $_n$ and **localCom** $_n$
- 16: remove chosen entry from **pendingAdds** $_{pchid}$
- 17: send (UPDATEADDHTLC, $pchid$, HTLCNo, $x, h, \text{CltvExpiry}, M$) to $pchid$
counterparty
- 18: Upon receiving (FULFILLONCHAIN) from \mathcal{E} :
- 19: send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign largest block number to t
- 20: **toSubmit** $\leftarrow \emptyset$
- 21: **for all** channels **do**
- 22: **if** there exists an HTLC in latest **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 $[t, t + (2 + r) \text{windowSize}]$
then
- 23: add **localCom** $_n$ of the channel and all corresponding valid
HTLC-successes and HTLC-timeouts (for both **localCom** $_n$ and **remoteCom** $_n$ ^a),
along with their signatures to **toSubmit**
- 24: **end if**
- 25: **end for**
- 26: send (SUBMIT, **toSubmit**) to $\mathcal{G}_{\text{Ledger}}$

^a Ensures funds retrieval if counterparty has gone on-chain

Fig. 32.

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 (**localHTLC_{n,i}**, *sh_{htlc,n}*)
- 10: add (sig, HTLCSigs_{*n,i*}, **localHTLC_{n,i}**) to HTLCs
- 11: **end for**
- 12: sig \leftarrow SIGNDS (**localCom_n**, *sh_F*)
- 13: add **receipt(channel)** to **closedChannels**
- 14: remove **channel** from **channels**
- 15: send (SUBMIT, (sig, **remoteSig_n**, **localCom_n**), HTLCs) to $\mathcal{G}_{\text{Ledger}}$

Fig. 33.

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 *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 *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: $sig \leftarrow \text{SIGNDS}(Cl, sk_{Alice})$
- 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 $\text{VERIFYDS}(\text{bobSig}, Cl, pk_{Bob}) = \text{True}$
- 27: $\text{aliceSig} \leftarrow \text{SIGNDS}(Cl, sk_{Alice})$
- 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. 34.