# 1 Blockchain & Network Anonymity Proposal

High-level proposal towards an application- and network-layer private fast payment system: There is an updateable CRS on-chain. When Alice wants to create a channel, she gets this CRS, updates it and gets a keypair and a new CRS as output. She publishes the new CRS on-chain – this transaction also transfers the money from her layer-1 address into a special address which contains all funds of the system's participants, so her anonymity set (from now on) is all other users of such channels. Since she hasn't disclosed yet the public key that was generated by the update-CRS procedure (a.k.a layer-2 address) and given that she waits for some time until she publishes it, her layer-2 address cannot be linked to her layer-1 address.

When she wants to send money to someone with which she has an open channel in the layer-2 system, she generates a SNARK with her secret key (as generated by the update-CRS process) that proves she pays the public key of the counterparty. She sends it to the recipient through a mixnet, using the layer-2 public key as address, so her network address cannot be linked to her layer-2 address.

When she wants to close the channel, she publishes the latest SNARK and takes her share of the funds from the special address. Timelocks are used as in Lightning to punish her if she publishes an old SNARK.

**Functionality** $\mathcal{F}_{\text{anonPayNet}}^{\mathcal{G}_{\text{Ledger}}}$

State: $\texttt{coins} : \mathcal{P} \to \mathbb{N}$, $\texttt{id} : \mathcal{P} \to \mathcal{ID}$, $(pk_\top, sk_\top)$

Upon receiving (REGISTER, $id$, $s$) from $P$:
  ensure $pk(s) = id$
  send (READ) to $\mathcal{G}_{\text{Ledger}}$ as $P$ and assign reply to $\Sigma_P$
  scan $\Sigma_P$ for UTXOs exclusively spendable by $id$ and assign the sum of their coins to $\texttt{coins}(P)$
  assign $(id, s)$ to $\texttt{id}(P)$
  create $\texttt{tx}$ which pays all funds spendable by $id$ to $pk_\top$ and contains new CRS (using $s$) TODO: refine
  send (SUBMIT, $\texttt{tx}$) to $\mathcal{G}_{\text{Ledger}}$ as $id$

Upon receiving (OPEN, $P_2$, $x$) from $P_1$:
  ensure $\texttt{coins}(P_1) \geq x$
  $chid \xleftarrow{\$} \{0,1\}^\lambda$
  $\texttt{coins}(P_1) \leftarrow \texttt{coins}(P_1) - x$
  $\texttt{coins}(chid)(P_1) \leftarrow x$
  $\texttt{coins}(chid)(P_2) \leftarrow 0$
  create special $\texttt{tx}$ based on most recent CRS and secret keys of $P_1$ and $P_2$. This $\texttt{tx}$ represents $chid$ TODO: refine
  send (SUBMIT, $\texttt{tx}$) to $\mathcal{G}_{\text{Ledger}}$ as $\texttt{id}(P_1)$

Upon receiving (PAY, $chid$, $x$) from $P_1$:
  **if** $\texttt{coins}(P_1) \geq x$ **then**
    $\texttt{coins}(chid)(P_1) \leftarrow \texttt{coins}(chid)(P_1) - x$
    $\texttt{coins}(chid)(P_2) \leftarrow \texttt{coins}(chid)(P_2) + x$
  **end if**
  send (NETWORK-ACTIVE) to $\mathcal{A}$

Upon receiving (CLOSE, $chid$) from $P_1$:
  ensure there is a channel $chid$ of which $P_1$ is a member
  create special $\texttt{tx}$ that, together with $\texttt{id}(P_1)_s$ or $\texttt{id}(P_2)_s$ can generate proof that $P_1$ had $\texttt{coins}(chid)(P_1)$ or $P_2$ had $\texttt{coins}(chid)(P_2)$ in channel $chid$ upon closure TODO: refine
  $\texttt{coins}(P_1) \leftarrow \texttt{coins}(P_1) + \texttt{coins}(chid)(P_1)$
  $\texttt{coins}(P_2) \leftarrow \texttt{coins}(P_2) + \texttt{coins}(chid)(P_2)$
  $\texttt{coins}(chid)(P_1) \leftarrow 0$
  $\texttt{coins}(chid)(P_2) \leftarrow 0$
  send (SUBMIT, $\texttt{tx}$) to $\mathcal{G}_{\text{Ledger}}$ as $\texttt{id}(P_1)$

Upon receiving (EXIT, $id$) from $P$:
  ensure $\texttt{id}(P) = id$
  create $\texttt{tx}$ which pays $\texttt{coins}(P)$ from $pk_\top$ to $id$ (using $sk_\top$)
  $\texttt{coins}(P) \leftarrow 0$
  send (SUBMIT, $\texttt{tx}$) to $\mathcal{G}_{\text{Ledger}}$ as $P$

**Fig. 1.** No privacy leakage

TODO:

## 2 Property Based approach to Payment Anonymity

**Game** Anonymous Payments $\mathcal{C}(\mathcal{P} : \{P_1 : c_1, \ldots, P_N : c_N\}, t \in \{1, \ldots, N\})$

State: Map $\mathcal{P} : P_i \rightarrow c_i$, mapping each player to its non-negative coins

Initialisation: $\texttt{max} \leftarrow \sum_{i=1}^{N} c_i; \texttt{stolen} \leftarrow 0; \texttt{failures} \leftarrow 0$

1: **while** $\texttt{failures} < t$ **do**
2: $P_i, P_j \xleftarrow{\$} \mathcal{P}; x \xleftarrow{\$} [0, \texttt{max}]$
3: **if** $x \leq c_i$ **then**
4: $c_j \leftarrow c_j + x; c_i \leftarrow c_i - x$
5: **end if**
6: send (PING) to $\mathcal{A}$
7: **if** response from $\mathcal{A}$ is (STEAL, $P_i$) : $P_i \in \mathcal{P}$ **then**
8: $\texttt{stolen} \leftarrow \texttt{stolen} + c_i$
9: $\texttt{failures} \leftarrow \texttt{failures} + 1$
10: remove $(P_i, c_i)$ from $\mathcal{P}$
11: **end if**
12: **end while**
13: **return** stolen

**Fig. 2.**

**Simulator** $\mathcal{A}_\$(\mathcal{P} : P_1, \ldots, P_N)$

1: Upon receiving (PING): **[Orfeas: Maybe send vector of players and steal all at once?]**
2: $P \xleftarrow{\$} \mathcal{P}$
3: remove $P$ from $\mathcal{P}$
4: send (STEAL, $P$)

**Fig. 3.**

**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*

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

## 3 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 $\bar{\mathcal{G}}_{\text{CLOCK}}$

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

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

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

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

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

**Fig. 4.**

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

1: Upon receiving (SEND, `party`, $M$) from *Alice*: // sender optional, can be part of payload
2: `id` $\xleftarrow{\$}$ $\{0,1\}^\lambda$
3: store (`party`, $M$, `id`)
4: send (NEW-MESSAGE, `id`) to $\mathcal{A}$

5: Upon receiving (CONTINUE, `id`) from $\mathcal{A}$:
6: retrieve (`party`, $M$, `id`) and remove entry from storage
7: send (PUSH, $M$) to `party`

**Fig. 5.**

## 4 Anonymous Communication Protocol

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

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

**Fig. 6.**

**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

Upon receiving (SEND, `receiver`, $M$) from $\mathcal{E}$:
    `player` $\xleftarrow{\$}$ $\mathcal{P}$
    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
    **if** `player` $\neq$ us **then**
        send (PASS-TOKEN, `player`) to $\mathcal{F}_{\text{token}}$
    **else**
        send (CIRCULATE, message of next player) to next player
    **end if**

Upon receiving (PASS-TOKEN) from $\mathcal{F}_{\text{token}}$:
    **if** `toEnv` = `True` **then**
        retrieve stored $M$ and send (PUSH, $M$) to $\mathcal{E}$
    **else**
        encrypt garbage for every player
        send (CIRCULATE, message of next player) to next player
    **end if**

Upon receiving (CIRCULATE, $C$) from *Bob*:
    decrypt $C$ to $M$
    **if** $M$ is `receiver` **then**
        store `receiver` for sending (PASS-TOKEN, `receiver`) to $\mathcal{F}_{\text{token}}$ upon receiving (CIRCULATE, $C'$) for the last time for this particular payload
    **else if** $M$ is a valid payload **then**
        store $M$
        `toEnv` $\leftarrow$ `True`
    **end if**
    send (CIRCULATE, encrypted garbage) to next player

**Fig. 7.** Realises $\mathcal{F}_{\text{anon}}$ in the global passive adversary case

## 5 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*, $x$, $\overrightarrow{\texttt{path}}$, `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*, $x$, $\overrightarrow{\texttt{path}}$, `receipt`, *payid*) - obsolete
- (CONTINUE) - obsolete
- (CLOSECHANNEL, *fchid*, *Alice*)
- (FORCECLOSECHANNEL, *fchid*, *Alice*)
- (POLL, $\Sigma_{Alice}$, *Alice*) - obsolete
- (PUSHFULFILL, *pchid*, *Alice*) - obsolete
- (PUSHADD, *pchid*, *Alice*) - obsolete
- (COMMIT, *pchid*, *Alice*) - obsolete
- (FULFILLONCHAIN, $t$, *Alice*) - obsolete

**Fig. 8.**

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

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

15: Upon receiving (TOPPEDUP) from *Alice*:
16: send (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and store reply to $\Sigma_{\mathit{Alice}}$
17: $\texttt{checkClosed}(\Sigma_{\mathit{Alice}})$
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. 9.**

**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*, $p_{Alice,F}$, $p_{Bob,F}$, *fchid*, *pchid*, *tid*) from $\mathcal{S}$:
7: ensure that there is a `pendingOpen`(*fchid*) entry with temporary id *tid*
8: add $p_{Alice,F}, p_{Bob,F}$, *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, $x, p_{Alice,F}, p_{Bob,F}$) $\leftarrow$ `pendingOpen`(*fchid*)
12: send (read) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and store reply to $\Sigma_{Alice}$
13: `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 `channel` with "waiting for fundingLocked"
16: send (fundingLocked, *Alice*, $\Sigma_{Alice}$, *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 $\Sigma_{Bob}$
20: `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 `receipt`(`channel`) to `newChannels`(*Bob*)
23: send (fundingLocked, *Bob*, $\Sigma_{Bob}$, *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. 10.**

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

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

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

3: starting on $|\overrightarrow{\texttt{path}}|$ clock ticks after receiving this message, on every clock tick, channel $\in \overrightarrow{\texttt{path}}$, 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 $\overrightarrow{\texttt{path}}$ and add `receipt` of new balance to both parties' `updatesToReport`, starting from the unique channel in which the payee is participating

**Fig. 11.**

**Functionality** $\mathcal{F}_{\text{PayNet}}$– 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`), $\infty$) 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`), $\perp$) 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`), $\infty$) from `pendingClose`(*Alice*)

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

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

**Fig. 12.**

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

**function** `checkClosed`($\Sigma_{Alice}$) // Called after every (READ), ensures requested closes eventually happen
  **if** there is any closing/commitment transaction in $\Sigma_{Alice}$ with no corresponding entry in `pendingClose`($Alice$) $\cup$ `closedChannels`($Alice$) **then**
    add ($fchid$, `receipt`, $\perp$) to `closedChannels`($Alice$), where $fchid$ is the ID of the corresponding `channel`, `receipt` comes from the latest `channel` state
  **end if**
  **for all** entries ($fchid$, `receipt`, $h$) $\in$ `pendingClose`($Alice$) $\cup$ `closedChannels`($Alice$) **do**
    **if** there is a closing/commitment transaction in $\Sigma_{Alice}$ for open `channel` with ID $fchid$ with a balance that corresponds to `receipt` **then**
      let $x, y$ $Alice$'s and `channel` counterparty $Bob$'s balances respectively
      offChainBalance ($Alice$) $\leftarrow$ offChainBalance ($Alice$) $- x$
      onChainBalance ($Alice$) $\leftarrow$ onChainBalance ($Alice$) $+ x$
      offChainBalance ($Bob$) $\leftarrow$ offChainBalance ($Bob$) $- y$
      onChainBalance ($Bob$) $\leftarrow$ onChainBalance ($Bob$) $+ y$
      remove `channel` from `channels` & entry from `pendingClose`($Alice$)
      **if** there is an ($fchid$, _, _) entry in `pendingClose`($Bob$) **then**
        remove it from `pendingClose`($Bob$)
      **end if**
    **else if** there is a tx in $\Sigma_{Alice}$ that is not a closing/commitment tx and spends the funding tx of the `channel` with ID $fchid$ **then**
      halt // DS forgery
    **else if** there is a commitment transaction in block of height $h$ in $\Sigma_{Alice}$ 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**
      **if** `polls`($Alice$) contains an entry in $[h, h + \texttt{delay}(Alice) - 1]$ **then**
        halt
      **else**
        `negligent`($Alice$) $\leftarrow$ `true`
      **end if**
    **else if** there is no such closing/commitment transaction $\wedge$ $h = \perp$ **then**
      assign largest block number of $\Sigma_{Alice}$ to $h$ of entry
    **else if** there is no such closing/commitment transaction $\wedge$ $h \neq \perp$ $\wedge$ (largest block number of $\Sigma_{Alice}$) $\geq h + (2 + r)$ `windowSize` **then**
      halt
    **end if**
  **end for**
  **if** $Alice$ has no open channels in $\Sigma_{Alice}$ AND `negligent`($Alice$) = `false` **then**
    **if** offChainBalance($Alice$) $\neq 0$ OR onChainBalance($Alice$) is not equal to the total funds exclusively spendable by $Alice$ in $\Sigma_{Alice}$ **then**
      halt
    **end if**
  **end if**
**end function**

**Fig. 13.**

**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 *fchid* and h from `closedChannels`(*Alice*)

**Fig. 14.**

– 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).

## 6 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** $\Pi_{\text{LN}}$ (self is *Alice* always) – support

Initialisation:
    $\texttt{channels}, \texttt{pendingOpen}, \texttt{pendingPay}, \texttt{pendingClose} \leftarrow \emptyset$
    $\texttt{newChannels}, \texttt{closedChannels}, \texttt{updatesToReport} \leftarrow \emptyset$
    $\texttt{unclaimedOfferedHTLCs}, \texttt{unclaimedReceivedHTLCs}, \texttt{pendingGetPaid} \leftarrow \emptyset$

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

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

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

**function** `GetKeys`
    $(p_F, s_F) \leftarrow$ KEYGEN () // For $F$ output
    $(p_{\text{pay}}, s_{\text{pay}}) \leftarrow$ SETUP () // For com output to remote
    $(p_{\text{dpay}}, s_{\text{dpay}}) \leftarrow$ SETUP () // For com output to self
    $(p_{\text{htlc}}, s_{\text{htlc}}) \leftarrow$ SETUP () // For htlc output to self
    $\texttt{seed} \xleftarrow{\$} U(k)$ // For per com point
    $(p_{\text{rev}}, s_{\text{rev}}) \leftarrow$ MASTERKEYGEN () // For revocation in com
    **return** $((p_F, s_F), (p_{\text{pay}}, s_{\text{pay}}), (p_{\text{dpay}}, s_{\text{dpay}}),$
        $(p_{\text{htlc}}, s_{\text{htlc}}), \texttt{seed}, (p_{\text{rev}}, s_{\text{rev}}))$
**end function**

**Fig. 15.**

**Protocol** $\Pi_{\text{LN}}$ – 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: $((ph_F, sh_F), (phb_{\text{pay}}, shb_{\text{pay}}), (phb_{\text{dpay}}, shb_{\text{dpay}}), (phb_{\text{htlc}}, shb_{\text{htlc}}), \texttt{seed}, (phb_{\text{rev}}, shb_{\text{rev}})) \leftarrow \texttt{GetKeys}()$
4: $\text{prand}_1 \leftarrow \text{PRF}(\texttt{seed}, 1)$
5: $(ph_{\text{com},1}, sh_{\text{com},1}) \leftarrow \text{KeyShareGen}(1^k; \text{prand}_1)$
6: associate keys with $tid$
7: add ($Alice$, $Bob$, $x$, $\texttt{tid}$, $(ph_F, sh_F)$, $(phb_{\text{pay}}, shb_{\text{pay}})$, $(phb_{\text{dpay}}, shb_{\text{dpay}})$ $(phb_{\text{htlc}}, shb_{\text{htlc}})$, $(phb_{\text{com},1}, shb_{\text{com},1})$, $(phb_{\text{rev}}, shb_{\text{rev}})$, $tid$) to $\texttt{pendingOpen}$
8: send (openChannel, $x, \texttt{delay} + (2 + r)\,\texttt{windowSize}, ph_F, phb_{\text{pay}}, phb_{\text{dpay}}, phb_{\text{htlc}}, ph_{\text{com},1}, phb_{\text{rev}}, tid$) to $Bob$

**Fig. 16.**

**Protocol** $\Pi_{\text{LN}}$ – openChannel from $Bob$

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

**Fig. 17.**

**Protocol** $\Pi_{\text{LN}}$ – acceptChannel

1: Upon receiving (acceptChannel, `remoteDelay`, $pt_F, ptb_{\text{pay}}, ptb_{\text{dpay}}, ptb_{\text{htlc}}, pt_{\text{com},1}, ptb_{\text{rev}}$, *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 $\Sigma_{Alice}$
5: assign to `prevout` a transaction output found in $\Sigma_{Alice}$ that is currently exclusively spendable by *Alice* and has value $y \geq x$
6: $F \leftarrow$ 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: $(x, ph_{\text{pay},1})$, $(0, ph_{\text{rev},1} \vee (pt_{\text{dpay},1}, \texttt{delay} + (2 + r)\ \texttt{windowSize}$ relative))}
14: `localCom` $\leftarrow$ TX {input: output 1 of $F$, outputs: $(x, pt_{\text{rev},1} \vee (ph_{\text{dpay},1}, \texttt{remoteDelay}$ relative)), $(0, pt_{\text{pay},1})$}
15: add `remoteCom` and `localCom` to channel entry in `pendingOpen`
16: sig $\leftarrow$ SignDS(`remoteCom`$_1, sh_F$)
17: `lastRemoteSigned` $\leftarrow 0$
18: send (fundingCreated, *tid*, *pchid*, sig) to *Bob*

**Fig. 18.**

**Protocol** $\Pi_{\text{LN}}$ – FUNDINGCREATED

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

**Fig. 19.**

**Protocol** $\Pi_{\text{LN}}$ – FUNDINGSIGNED

1: Upon receiving (FUNDINGSIGNED, *pchid*, $\texttt{BobSig}_1$) from *Bob*:
2: ensure there is a channel ID *pchid* with *Bob* in `pendingOpen` on which we have sent up to FUNDINGCREATED
3: ensure $\text{VERIFYDS}\,(\texttt{BobSig}_1, \texttt{localCom}, pb_F) = \texttt{True}$
4: $\texttt{localCom}_1 \leftarrow \texttt{localCom}$
5: $\texttt{lastLocalSigned} \leftarrow 0$
6: add $\texttt{BobSig}_1$ 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 $\mathcal{G}_{\text{Ledger}}$

**Fig. 20.**

**Protocol** $\Pi_{\text{LN}}$ – checkForNew

Upon receiving (checkForNew, *Alice*, *Bob*, *tid*) from $\mathcal{E}$: // lnd polling daemon
    ensure there is a matching `channel` in `pendingOpen` with id *pchid*, with a "broadcast" and a "no fundingLocked" mark, funded with $x$ coins
    send (read) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to $\Sigma_{Alice}$
    ensure $\exists$ unspent TX in $\Sigma_{Alice}$ with ID *pchid* and a $(x, ph_F \wedge pt_F)$ output
    $\text{prand}_2 \leftarrow \text{PRF}(\texttt{seed}, 2)$
    $(ph_{\text{com},2}, sh_{\text{com},2}) \leftarrow \text{KeyShareGen}\left(1^k; \text{prand}_2\right)$
    add TX to `channel` data
    replace "broadcast" mark in `channel` with "fundingLocked sent"
    send (fundingLocked, *pchid*, $ph_{\text{com},2}$) to *Bob*

**Fig. 21.**

**Protocol** $\Pi_{\text{LN}}$ – fundingLocked

Upon receiving (fundingLocked, *pchid*, $pt_{\text{com},2}$) from *Bob*:
    ensure there is a `channel` with ID *pchid* with *Bob* in `pendingOpen` with a "no fundingLocked" mark
    **if** `channel` is not marked with "fundingLocked sent" **then** // i.e. marked with "broadcast"
        send (read) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to $\Sigma_{Alice}$
        ensure $\exists$ unspent TX in $\Sigma_{Alice}$ with ID *pchid* and a $(x, ph_F \wedge pt_F)$ output
        add TX to `channel` data
        $\text{prand}_2 \leftarrow \text{PRF}(\texttt{seed}, 2)$
        $(ph_{\text{com},2}, sh_{\text{com},2}) \leftarrow \text{KeyShareGen}\left(1^k; \text{prand}_2\right)$
        generate 2nd remote delayed payment, htlc, payment keys
    **end if**
    replace "no fundingLocked" mark in `channel` with "fundingLocked received"
    move channel data from `pendingOpen` to `channels`
    add receipt of channel to `newChannels`, where receipt $\leftarrow (Alice : x, Bob : 0, pchid)$
    **if** `channel` is not marked with "fundingLocked sent" **then**
        replace "broadcast" mark in `channel` with "fundingLocked sent"
        send (fundingLocked, *pchid*, $ph_{\text{com},2}$) to *Bob*
    **end if**

**Fig. 22.**

**Protocol** $\Pi_{\text{LN}}$ – poll

Upon receiving (POLL) from $\mathcal{E}$:
  send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to $\Sigma_{Alice}$
  `toSubmit` $\leftarrow \emptyset$
  **for all** $\tau \in$ `unclaimedOfferedHTLCs` **do**
    **if** input of $\tau$ has been spent **then** // by `remote HTLC-success`
      remove $\tau$ from `unclaimedOfferedHTLCs`
      **if** we are intermediary **then**
        retrieve preimage $R$, $pchid'$ of previous channel on the path of the HTLC, and `HTLCNo`$'$ of the corresponding HTLC$'$ in $pchid'$
        add (`HTLCNo`$', R$) to $\texttt{pendingFulfills}_{pchid'}$
      **end if**
    **else if** input of $\tau$ has not been spent and timelock is over **then**
      remove $\tau$ from `unclaimedOfferedHTLCs`
      add $\tau$ to `toSubmit`
    **end if**
  **end for**
  run loop of Fig. **??**
  **for all** honestly closed $\texttt{remoteCom}_n$ that were processed above, with channel id $pchid$ **do**
    **for all** received HTLC outputs $i$ of $\texttt{remoteCom}_n$ **do**
      **if** there is an entry in $\texttt{pendingFulfills}_{pchid}$ with the same `HTLCNo` and $R$ **then**
        TX $\leftarrow$ {input: $i$ HTLC output of $\texttt{remoteCom}_n$ with $(ph_{\text{htlc},n}, R)$ as method, output: $pk_{Alice}$}
        sig $\leftarrow$ SIGNIBS $(TX, sh_{\text{htlc},n})$
        add (sig, TX) to `toSubmit`
        remove entry from $\texttt{pendingFulfills}_{pchid}$
      **end if**
    **end for**
  **end for**
  send (SUBMIT, `toSubmit`) to $\mathcal{G}_{\text{Ledger}}$

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

**Fig. 23.**

Loop over closed channels for poll

**for all** $\texttt{remoteCom}_n \in \Sigma_{Alice}$ that spend $F$ of a $\texttt{channel} \in \texttt{channels}$ **do**
  **if** we do not have $sh_{\text{rev},n}$ **then** // Honest closure
    **for all** unspent offered HTLC outputs $i$ of $\texttt{remoteCom}_n$ **do**
      TX $\leftarrow$ {input: $i$ HTLC output of $\texttt{remoteCom}_n$ with $ph_{\text{htlc},n}$ as method, output: $pk_{Alice}$}
      sig $\leftarrow$ SIGNIBS $(TX, sh_{\text{htlc},n})$
      **if** timelock has not expired **then**
        add (sig, TX) to `unclaimedOfferedHTLCs`
      **else if** timelock has expired **then**
        add (sig, TX) to `toSubmit`
      **end if**
    **end for**
    **for all** spent offered HTLC output $i$ of $\texttt{remoteCom}_n$ **do**
      **if** we are intermediary **then**
        retrieve preimage $R$, $pchid'$ of previous channel on the path of the HTLC, and $\texttt{HTLCNo}'$ of the corresponding HTLC$'$ in $pchid'$
        add $(\texttt{HTLCNo}', R)$ to $\texttt{pendingFulfills}_{pchid'}$
      **end if**
    **end for**
  **else** // malicious closure
    rev $\leftarrow$ TX {inputs: all $\texttt{remoteCom}_n$ outputs, choosing $ph_{\text{rev},n}$ method, output: $pk_{Alice}$}
    sig $\leftarrow$ SIGNCS $(\text{rev}, sh_{\text{rev},n})$
    add (sig, rev) to `toSubmit`
  **end if**
  add `receipt(channel)` to `closedChannels`
  remove `channel` from `channels`
**end for**

**Fig. 24.**

**Protocol** $\Pi_{\text{LN}}$ – 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. 25.**

**Protocol** $\Pi_{\text{LN}}$ – invoice

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

**Fig. 26.**

**Protocol** $\Pi_{\text{LN}}$ – updateAddHtlc

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

**Fig. 27.**

**Protocol** $\Pi_{\text{LN}}$ – updateAddHtlc checks

ensure *pchid* corresponds to an open `channel` in `channels` where *Bob* has at least $x$
ensure $\texttt{HTLCNo} = \texttt{HTLCNo}_{pchid} + 1$
$(pchid', x', \texttt{OutgoingCltvExpiry}, \delta) \leftarrow \texttt{SphinxPeel}\,(sk_{Alice}, M)$
send (read) to $\mathcal{G}_{\text{Ledger}}$ and assign largest block number to $t$
**if** $\delta = \texttt{receiver}$ **then**
    ensure $pchid' = \bot, x = x', \texttt{IncomingCltvExpiry} \geq \texttt{OutgoingCltvExpiry} = \texttt{minFinalCltvExpiry}$
    mark `HTLCNo` as "receiver"
**else** // We are an intermediary
    ensure $x = x', \texttt{IncomingCltvExpiry} \geq \max\{\texttt{OutgoingCltvExpiry}, t\} + \texttt{relayDelay} + 2\,(2 + r)\,\texttt{windowSize}$
    ensure $\texttt{pchid}'$ corresponds to an open channel in `channels` where we have at least $x$
    mark `HTLCNo` as "intermediary"
**end if**

**Fig. 28.**

**Protocol** $\Pi_{\text{LN}}$ – 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 $\mathcal{G}_{\text{Ledger}}$ and assign reply to $\Sigma_{Alice}$
13: **if** latest $\texttt{remoteCom}'_n \in \Sigma_{Alice}$ **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, $sh_{\text{htlc},n}$)
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, $pchid'$, `HTLCNo`$'$, $R$) to counterparty
20: **end if**
21: **end if**

**Fig. 29.**

**Protocol** $\Pi_{\text{LN}}$ – COMMIT

Upon receiving (COMMIT, *pchid*) from $\mathcal{E}$:
  ensure that there is a `channel` $\in$ `channels` with ID *pchid*
  retrieve latest remote commitment tx $\texttt{remoteCom}_n$ in `channel`
  ensure `remoteCom` $\neq \texttt{remoteCom}_n$ // there are uncommitted updates
  ensure `channel` is not marked as "waiting for REVOKEANDACK"
  send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign largest block number to $t$
  undo adding all outgoing HTLCs in `remoteCom` for which we are intermediary and `IncomingCltvExpiry` $< t +$ `relayDelay` $+ (2 + r)$ `windowSize`
  $\texttt{remoteCom}_{n+1} \leftarrow$ `remoteCom`
  ComSig $\leftarrow$ SIGNDS $(\texttt{remoteCom}_{n+1}, sh_F)$
  HTLCSigs $\leftarrow \emptyset$
  **for** $i$ from `lastRemoteSigned` + 1 to `HTLCNo` **do**
    $\texttt{remoteHTLC}_{n+1,i} \leftarrow$ TX {input: HTLC output $i$ of $\texttt{remoteCom}_{n+1}$, output: ($c_{\text{htlc,i}}$, $ph_{\text{rev},n+1} \vee (pt_{\text{dpay},n+1}$, `delay` $+ (2 + r)$ `windowSize` relative))}
    add SIGNIBS $(\texttt{remoteHTLC}_{n+1,i}, sh_{\text{htlc},n+1})$ to HTLCSigs
  **end for**
  `lastRemoteSigned` $\leftarrow$ `HTLCNo`
  mark `channel` as "waiting for REVOKEANDACK"
  send (COMMITMENTSIGNED, *pchid*, ComSig, HTLCSigs) to *pchid* counterparty

**Fig. 30.**

**Protocol** $\Pi_{\text{LN}}$ – commitmentSigned

1: Upon receiving (commitmentSigned, *pchid*, $\texttt{comSig}_{n+1}$, $\texttt{HTLCSigs}_{n+1}$) from *Bob*:
2: ensure that there is a `channel` $\in$ `channels` with ID *pchid* with *Bob*
3: retrieve latest local commitment tx $\texttt{localCom}_n$ in `channel`
4: ensure `localCom` $\neq \texttt{localCom}_n$ and `localCom` $\neq$ `pendingLocalCom` // there are uncommitted updates
5: **if** VerifyDS $\left(\texttt{comSig}_{n+1}, \texttt{localCom}, pt_F\right) = \texttt{false} \vee |\texttt{HTLCSigs}_{n+1}| \neq$ `HTLCNo` $-$ `lastLocalSigned` **then**
6: close channel (as in Fig. **??**)
7: **return**
8: **end if**
9: **for** $i$ from `lastLocalSigned` + 1 to `HTLCNo` **do**
10: $\texttt{localHTLC}_{n+1,i} \leftarrow$ TX {input: HTLC output $i$ of `localCom`, output: ($c_{\text{htlc,i}}$, $ph_{\text{rev},n+1} \vee (pt_{\text{dpay},n+1}$, `remoteDelay` relative))}
11: **if** VerifyIBS($\texttt{HTLCSigs}_{n+1,i}$, $\texttt{localHTLC}_{n+1,i}$, $pt_{\text{htlc},n+1}$) = `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: $\text{prand}_{n+2} \leftarrow \text{PRF}\left(\texttt{seed}, n+2\right)$
20: $(ph_{\text{com},n+2}, sh_{\text{com},n+2}) \leftarrow$ KeyShareGen $\left(1^k; \text{prand}_{n+2}\right)$
21: send (revokeAndAck, *pchid*, $\text{prand}_n, ph_{\text{com},n+2}$) to *Bob*

**Fig. 31.**

**Protocol** $\Pi_{\text{LN}}$ – REVOKEANDACK

1: Upon receiving (REVOKEANDACK, *pchid*, $st_{\text{com},n}, pt_{\text{com},n+2}$) from *Bob*:
2: ensure there is a `channel` ∈ `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 $\texttt{remoteCom}_{n+1}$ as "irrevocably committed"
8: $\texttt{localCom}_{n+1} \leftarrow \texttt{pendingLocalCom}$
9: unmark `channel`
10: add `receipt`(`channel`) to `updatesToReport`
11: $sh_{\text{rev},n} \leftarrow \text{COMBINEKEY}\,(phb_{\text{rev}}, shb_{\text{rev}}, pt_{\text{comn}}, st_{\text{com},n})$
12: $ph_{\text{rev},n+2} \leftarrow \text{COMBINEPUBKEY}\,(phb_{\text{rev}}, pt_{\text{com},n+2})$
13: $pt_{\text{rev},n+2} \leftarrow \text{COMBINEPUBKEY}\,(ptb_{\text{rev}}, ph_{\text{com},n+2})$
14: $(ph_{\text{dpay},n+2}, sh_{\text{dpay},n+2}) \leftarrow \text{KEYDER}\,(phb_{\text{dpay}}, shb_{\text{dpay}}, ph_{\text{com},n+2})$
15: $pt_{\text{dpay},n+2} \leftarrow \text{PUBKEYDER}\,(ptb_{\text{dpay}}, pt_{\text{com},n+2})$
16: $(ph_{\text{pay},n+2}, sh_{\text{pay},n+2}) \leftarrow \text{KEYDER}\,(phb_{\text{pay}}, shb_{\text{pay}}, ph_{\text{com},n+2})$
17: $pt_{\text{pay},n+2} \leftarrow \text{PUBKEYDER}\,(ptb_{\text{pay}}, pt_{\text{com},n+2})$
18: $(ph_{\text{htlc},n+2}, sh_{\text{htlc},n+2}) \leftarrow \text{KEYDER}\,(phb_{\text{htlc}}, shb_{\text{htlc}}, ph_{\text{com},n+2})$
19: $pt_{\text{htlc},n+2} \leftarrow \text{PUBKEYDER}\,(ptb_{\text{htlc}}, pt_{\text{com},n+2})$
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. 32.**

**Protocol** $\Pi_{\text{LN}}$ – 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 $\texttt{pendingFulfills}_{pchid}$ that is both in an "irrevocably committed" $\texttt{remoteCom}_n$ and $\texttt{localCom}_n$
4: send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to $\Sigma_{Alice}$
5: remove (`HTLCNo`, $R$) from $\texttt{pendingFulfills}_{pchid}$
6: **if** $\texttt{remoteCom}_n \notin \Sigma_{Alice}$ **then** // counterparty cooperative
7: send (updateFulfillHtlc, *pchid*, `HTLCNo`, $R$) to *pchid* counterparty
8: **else** // counterparty gone on-chain
9: TX $\leftarrow$ {input: ($\texttt{remoteCom}_n$ 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 ($\texttt{HTLCNo}, x, h, \texttt{CltvExpiry}, M$) of $\texttt{pendingAdds}_{pchid}$ that is both in an "irrevocably committed" $\texttt{remoteCom}_n$ and $\texttt{localCom}_n$
16: remove chosen entry from $\texttt{pendingAdds}_{pchid}$
17: send (updateAddHtlc, *pchid*, $\texttt{HTLCNo}, x, h, \texttt{CltvExpiry}, M$) 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 $\texttt{localCom}_n$ 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 $[t, t + (2 + r)\,\texttt{windowSize}]$ **then**
23: add $\texttt{localCom}_n$ of the channel and all corresponding valid `HTLC-success`es and `HTLC-timeout`s (for both $\texttt{localCom}_n$ and $\texttt{remoteCom}_n$[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. 33.**

**Protocol** $\Pi_{\text{LN}}$ – close unilaterally

Upon receiving (FORCECLOSECHANNEL, `receipt`, *pchid*) from $\mathcal{E}$:
  ensure `receipt` corresponds to an open `channel` $\in$ `channels` with ID *pchid*
  **if** the sequence for CLOSECHANNEL has been initiated and is pending on clearing all outstanding HTLCs **then**
    forget this "hook"
  **end if**
  assign latest `channel` sequence number to $n$
  HTLCs $\leftarrow \emptyset$
  **for** every HTLC output $\in$ $\texttt{localCom}_n$ with number $i$ **do**
    sig $\leftarrow$ SIGNIBS $(\texttt{localHTLC}_{n,i}, sh_{\text{htlc},n})$
    add $\left(\text{sig}, \texttt{HTLCSigs}_{n,i}, \texttt{localHTLC}_{n,i}\right)$ to HTLCs
  **end for**
  sig $\leftarrow$ SIGNDS $(\texttt{localCom}_n, sh_F)$
  add `receipt`(`channel`) to `closedChannels`
  remove `channel` from `channels`
  send (SUBMIT, $(\text{sig}, \texttt{remoteSig}_n, \texttt{localCom}_n)$, HTLCs) to $\mathcal{G}_{\text{Ledger}}$

**Fig. 34.**

**Protocol** $\Pi_{\text{LN}}$ – close cooperatively

Upon receiving (CLOSECHANNEL, `receipt`, *pchid*) from $\mathcal{E}$:
  ensure `receipt` corresponds to an open `channel` $\in$ `channels` with ID *pchid*
  stop serving any (PAY, CLOSECHANNEL) message from $\mathcal{E}$ for this channel.
  mark `channel` as "coop closing"
  **if** there are outstanding HTLC outputs in the latest $\texttt{localCom}_n$ **then** continue from here when there are none left
  **end if**
  send (SHUTDOWN, $pk_{Alice}$, *pchid*) to *Bob*

Upon receiving (SHUTDOWN, $pk_{Bob}$, *pchid*) from *Bob*:
  ensure there is an open `channel` $\in$ `channels` with *Bob* with ID *pchid*
  **if** `channel` is not marked "coop closing" **then**
    mark channel as "coop closing"
    **if** there are outstanding HTLC outputs in the latest $\texttt{localCom}_n$ **then** continue from here when there are none left
    **end if**
    send (SHUTDOWN, $pk_{Alice}$, *pchid*) to *Bob*
  **else**
    $Cl \leftarrow$ 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
    sig $\leftarrow$ SIGNDS($Cl, sk_{Alice}$)
    send (CLOSINGSIGNED, sig, *pchid*) to *Bob*
  **end if**

Upon receiving (CLOSINGSIGNED, bobSig, *pchid*) from *Bob*:
  ensure there is an open `channel` $\in$ `channels` with *Bob* with ID *pchid*
  ensure `channel` is marked as "coop closing"
  add `receipt`(`channel`) to `closedChannels`
  remove `channel` from `channels`
  $Cl \leftarrow$ 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
  ensure VERIFYDS(bobSig, $Cl, pk_{Bob}$) = `True`
  aliceSig $\leftarrow$ SIGNDS($Cl, sk_{Alice}$)
  sort aliceSig, bobSig according to the ordering of the respective keys to produce sig1, sig2
  send (SUBMIT, ((sig1, sig2), $Cl$)) to $\mathcal{G}_{\text{Ledger}}$

**Fig. 35.**