Payment Channels Overview

Orfeas Stefanos Thyfronitis Litos

University of Edinburgh
o.thyfronitis@ed.ac.uk

Abstract. We provide a payment network functionality and prove that the Lightning Network [1] UC-realizes it.

1 State of a channel

Consider a channel between *Alice* and *Bob*. Both parties hold some data locally that correspond to ownership of some funds in the channel. Here we define a concise way of representing this data.

What *Alice* has to hold, specific for this channel:

- keys:
  - local funding secret key
  - remote funding public key
  - local {payment, htlc, delayed_payment, revocation}_basepoint_secret
  - remote {payment, htlc, delayed_payment, revocation}_basepoint
  - seed (for local per_commitment_secrets)
  - remote per_commitment_secret$_{1,\ldots,m-1}$
  - remote per_commitment_point$_{m,m+1}$
- *Alice*'s coins
- *Bob*'s coins
- every HTLC that is included in the latest irrevocably committed (local or remote) commitment:
  - direction (*Alice* $\rightarrow$ *Bob* or *Bob* $\rightarrow$ *Alice*)
  - hash
  - preimage (or $\perp$ if still unresolved)
  - coins
  - Is it included in local commitment$_n$?
  - HTLC number
- signatures:
  - signature of local commitment$_n$ with secret key corresponding to remote funding public key

- for every HTLC included in local commitment$_n$, one signature of HTLC-Timeout if outgoing, HTLC-Success if incoming with secret key corresponding to remote htlc_pubkey$_n$ (= htlc_basepoint + $\mathcal{H}$(remote per_commitment_point$_n$ ||remote htlc_basepoint)$\cdot G$)

The rest of the things used in the protocol can be derived by the above.

Representation of a channel's state (from the point of view of *Alice*):

- *Alice*'s coins c_{Alice}
- *Bob*'s coins c_{Bob}
- list of (coins, state $\in \{\text{proposed}, \text{committed}\}$) preimage, whether we have a signature), `HTLCs`
  - negative coins are outgoing, positive are incoming
  - HTLCs can either be simply proposed (not in an irrevocably committed remote transaction) or committed (the opposite). After the preimage is supplied (no matter the direction), the HTLC is considered settled and is discarded.

I.e. $\text{State}_{Alice,pchid} = (c_{Alice}, c_{Bob}, ((c_1, \text{state}_1), \ldots, (c_k, \text{state}_k)))$
E.g. $\text{State}_{Alice,pchid} = (4, 5, ((0.1, \text{proposed}), (-0.2, \text{signed})))$
We do not include in the state elements whose contents are irrelevant (e.g. sigs, keys, hashes).

2 UC conventions

- send (READ) to $\mathcal{G}_{\text{Ledger}}$ and assign reply to Σ ...
  =
  {
  send (READ) to $\mathcal{G}_{\text{Ledger}}$

  upon receiving delayed output Σ ...
  }
- every output that is returned by $\mathcal{F}_{\text{PayNet}}$ or a player to $\mathcal{E}$ is in fact a delayed output: It is handed over to $\mathcal{A}$, who in turn decides when to give it to $\mathcal{E}$.

3 Differences from LND

- They use an ad-hoc construction for generating progressive secrets from seed and index, we use a PRF.

- To generate several public keys from one piece of info, they use the basepoint and the per commitment point and take advantage of EC homomorphic properties. We use an Identity Based Signature scheme.
- They also provide a way to cooperatively close a channel.
- In LND there are more messages that cover errors in transmission etc. There are also rules that govern message retransmission upon connection failure.

4 Transaction Structure

A well-formed transaction contains:

- A list of inputs
- A list of outputs
- An arbitrary payload (optional)

Each input must be connected to a single valid, previously unconnected (unspent) output in the state.

We assume a one-way, collision-free hash function $\mathcal{H}$ that creates the id of each transaction.

A well-formed output contains:

- A value in coins
- A list of spending methods. An input that spends this output must specify exactly one of the available spending methods.

A well-formed spending method contains any combination of the following:

- Public keys in disjunctive normal form. An input that spends using this spending method must contain signatures made with the private keys that correspond to the public keys of one of the conjunctions. If empty, no signatures are needed.
- Absolute locktime in block height, transaction height or time. The output can be spent by an input to a transaction that is added to the state after the specified block height, transaction height or time.
- Relative locktime in block height, transaction height or time. The output can be spent by an input that is added to the state after the current output has been part of the state for the specified number of blocks, transactions or time.
- Hashlock value. The output can be spent by an input that contains a preimage that hashes to the hashlock value. If empty, the input does not need to specify a preimage.

If both the absolute and the relative locktime are empty, output can be spent immediately after being added to the state.

A well-formed input contains:

- A reference to the output and the spending method it spends
- A set of signatures that correspond to one of the conjunctions of public keys in the referred spending method (if needed)
- A preimage that hashes to the hashlock value of the referred spending method (if needed)

Lastly, the sum of coins of the outputs referenced by the inputs of the transaction (to-be-spent outputs) should be greater than or equal to the sum of coins of the outputs of the transaction.

We say that an unspent output is currently exclusively spendable by a player *Alice* with a public key pk and a hash list hl if for each spending method one of the following two holds:

- It still has a locktime that has not expired and thus is currently unspendable, or
- The only specified public key is pk and if there is a hashlock, its hash is contained in hl.

If an output is exclusively spendable, we say that its coins are exclusively spendable.

5 Lightning Protocol

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

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

Upon receiving (REGISTER, delay, relayDelay) from $\mathcal{E}$:
  $\texttt{delay} \leftarrow$ delay
  $\texttt{relayDelay} \leftarrow$ relayDelay
  send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign largest block number to $\texttt{lastPoll}$
  $(pk_{Alice}, sk_{Alice}) \leftarrow \mathrm{KeyGen}()$
  send (REGISTER, *Alice*, $\texttt{delay}$, $\texttt{relayDelay}$, pk_{Alice}) to $\mathcal{E}$

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

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

function GETKEYS change font
  $(p_F, s_F) \leftarrow \mathrm{KeyGen}()$ // For F output
  $(p_{\mathrm{pay}}, s_{\mathrm{pay}}) \leftarrow \mathrm{MKeyGen}()$ // For com output to remote
  $(p_{\mathrm{dpay}}, s_{\mathrm{dpay}}) \leftarrow \mathrm{MKeyGen}()$ // For com output to self
  $(p_{\mathrm{htlc}}, s_{\mathrm{htlc}}) \leftarrow \mathrm{MKeyGen}()$ // For htlc output to self
  $\texttt{seed} \xleftarrow{\$} U(k)$ // For per com point
  $(p_{\mathrm{rev}}, s_{\mathrm{rev}}) \leftarrow \mathrm{MKeyGen}()$ // For revocation in com
  return $((p_F, s_F), (p_{\mathrm{pay}}, s_{\mathrm{pay}}), (p_{\mathrm{dpay}}, s_{\mathrm{dpay}}),$
    $(p_{\mathrm{htlc}}, s_{\mathrm{htlc}}), \texttt{seed}, (p_{\mathrm{rev}}, s_{\mathrm{rev}}))$
end function

Fig. 1.

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

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

Fig. 2.

Protocol Π_{LN} - OPENCHANNEL from Bob

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

Fig. 3.

Protocol Π_{LN} - ACCEPTCHANNEL

Upon receiving (ACCEPTCHANNEL, `remoteDelay`, $pt_F, ptb_{\mathrm{pay}}, ptb_{\mathrm{dpay}}$, ptb_{htlc}, $pt_{\mathrm{com},1}, ptb_{\mathrm{rev}}$, *tid*) from *Bob*:

  ensure there is a temporary ID *tid* with *Bob* in `pendingOpen` on which ACCEPTCHANNEL hasn't been received

  associate received keys with *tid*

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

  assign to `prevout` a transaction output found in Σ_{Alice} that is currently exclusively spendable by *Alice* and has value $y \geq x$

  $F \leftarrow$ TX {input spends prevout with a `signature`(TX, sk_{Alice}), output 0 pays $y - x$ to pk_{Alice}, output 1 pays x to $tid.ph_F \wedge pt_F$}

  $pchid \leftarrow \mathcal{H}(F)$

  add *pchid* to `pendingOpen` entry with id *tid*

  $pt_{\mathrm{rev},1} \leftarrow$ `CombinePubKey` $(ptb_{\mathrm{rev}}, ph_{\mathrm{com},1})$

  $ph_{\mathrm{dpay},1} \leftarrow$ `PubKeyGen` $(phb_{\mathrm{dpay}}, ph_{\mathrm{com},1})$

  $ph_{\mathrm{pay},1} \leftarrow$ `PubKeyGen` $(phb_{\mathrm{pay}}, ph_{\mathrm{com},1})$

  `remoteCom` $\leftarrow$ `remoteCom`$_1 \leftarrow$ TX {input: output 1 of F, output: $(x, ph_{\mathrm{pay},1})$}

  `localCom` $\leftarrow$ TX {input: output 1 of F, output: $(x, pt_{\mathrm{rev},1} \vee (ph_{\mathrm{dpay}_1},$ `remoteDelay` $+ k + 1$ relative))

  add `remoteCom` and `localCom` to channel entry in `pendingOpen`

  sig $\leftarrow$ `signature` (`remoteCom`$_1, sh_F$)

  `lastRemoteSigned` $\leftarrow 0$

  send (FUNDINGCREATED, *tid*, *pchid*, sig) to *Bob*

Fig. 4.

Protocol Π_{LN} - $\textsc{fundingCreated}$

Upon receiving ($\textsc{fundingCreated}$, *tid*, *pchid*, $\texttt{BobSig}_1$) from *Bob*:
    ensure there is a temporary ID *tid* with *Bob* in $\texttt{pendingOpen}$ on which we have sent up to $\textsc{acceptChannel}$
    $ph_{\text{rev},1} \leftarrow \texttt{CombinePubKey}\left(phb_{\text{rev}} pt_{\text{com},1}\right)$
    $pt_{\text{dpay},1} \leftarrow \texttt{PubKeyGen}\left(ptb_{\text{dpay}}, pt_{\text{com},1}\right)$
    $pt_{\text{pay},1} \leftarrow \texttt{PubKeyGen}\left(ptb_{\text{pay}}, pt_{\text{com},1}\right)$
    $\texttt{localCom} \leftarrow \texttt{localCom}_1 \leftarrow \text{TX}\left\{\text{input: output 1 of } F, \text{ output: } (x, pt_{\text{pay},1})\right\}$
    ensure $\texttt{verify}\left(\texttt{localCom}_1, \texttt{BobSig}_1, pt_F\right) = \texttt{True}$
    $\texttt{remoteCom} \leftarrow \texttt{remoteCom}_1 \leftarrow \text{TX}\left\{\text{input: output 1 of } F, \text{ output: } (x, ph_{\text{rev}} \vee (pt_{\text{dpay}}, \texttt{delay} + k + 1 \text{ relative}))\right\}$
    add $\texttt{BobSig}_1, \texttt{remoteCom}_1$ and $\texttt{localCom}_1$ to channel entry in $\texttt{pendingOpen}$
    $\text{sig} \leftarrow \texttt{signature}\left(\text{remoteCom}_1, sh_F\right)$
    mark channel as "broadcast, no $\textsc{fundingLocked}$"
    $\texttt{lastRemoteSigned}, \texttt{lastLocalSigned} \leftarrow 0$
    send ($\textsc{fundingSigned}$, *pchid*, sig) to *Bob*

Fig. 5.

Protocol Π_{LN} - $\textsc{fundingSigned}$

Upon receiving ($\textsc{fundingSigned}$, *pchid*, $\texttt{BobSig}_1$) from *Bob*:
    ensure there is a channel ID *pchid* with *Bob* in $\texttt{pendingOpen}$ on which we have sent up to $\textsc{fundingCreated}$
    ensure $\texttt{verify}\left(\texttt{localCom}, \texttt{BobSig}_1, pb_F\right) = \texttt{True}$
    $\texttt{localCom}_1 \leftarrow \texttt{localCom}$
    $\texttt{lastLocalSigned} \leftarrow 0$
    add $\texttt{BobSig}_1$ to channel entry in $\texttt{pendingOpen}$
    $\text{sig} \leftarrow \texttt{signature}\left(F, sk_{Alice}\right)$
    mark *pchid* in $\texttt{pendingOpen}$ as "broadcast, no $\textsc{fundingLocked}$"
    send ($\textsc{submit}$, (sig, F)) to $\mathcal{G}_{\text{Ledger}}$

Fig. 6.

Protocol Π_{LN} - CHECKNEW

explicitly add keys et al to `channel`
Upon receiving (CHECKNEW, *Alice*, *Bob*, *tid*) from $\mathcal{E}$: // new message: represents lnd polling daemon
    ensure there is a matching `channel` in `pendingOpen` with id *pchid*, with a "broadcast" mark, funded with x coins
    send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}
    ensure $\exists$ unspent TX in Σ_{Alice} with ID *pchid* and a $(x, ph_F \wedge pt_F)$ output
    $\mathrm{prand}_2 \leftarrow \texttt{PRF}\left(\texttt{seed}, 2\right)$
    $(sh_{\mathrm{com},2}, ph_{\mathrm{com},2}) \leftarrow \mathrm{KeyShareGen}\left(1^k; \mathrm{prand}_2\right)$
    add TX to `channel` data
    replace "broadcast" mark in `channel` with "in state"
    if `channel` is marked as "in state, FUNDINGLOCKED" **then**
        move channel data from `pendingOpen` to `channels`
        add receipt of channel to `newChannels`
    end if
    send (FUNDINGLOCKED, *pchid*, $ph_{\mathrm{com},2}$) to *Bob*

Fig. 7.

Protocol Π_{LN} - FUNDINGLOCKED

Upon receiving (FUNDINGLOCKED, *pchid*, $pt_{\mathrm{com},2}$) from *Bob*:
    ensure there is a `channel` with ID *pchid* with *Bob* in `pendingOpen` with a "no FUNDINGLOCKED" mark
    ensure $pk\left(st_{\mathrm{com},n}\right) = pt_{\mathrm{com},n}$
    replace "no FUNDINGLOCKED" mark in `channel` with "FUNDINGLOCKED"
    ensure `channel` has an "in state" mark
    generate 2nd remote delayed payment, htlc, payment keys
    add TX to `channel` data
    move channel data from `pendingOpen` to `channels`
    add receipt of channel to `newChannels`

Fig. 8.

Protocol $\varPi_{\mathrm{LN}}$ - poll

Upon receiving (POLL) from $\mathcal{E}$:
    send (READ) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}
    assign largest block number in Σ_{Alice} to `lastPoll`
    `toSubmit` $\leftarrow \emptyset$
    for all $\tau \in$ `unclaimedOfferedHTLCs` **do**
        if input of τ has been spent **then** // by `remote HTLC-success`
            remove τ from `unclaimedOfferedHTLCs`
            remember preimage - hash combination
        else if input of τ has not been spent and timelock is over **then**
            remove τ from `unclaimedOfferedHTLCs`
            add τ to `toSubmit`
        end if
    end for
    for all `remoteCom`$_n \in \Sigma_{Alice}$ that spend F of a `channel` $\in$ `channels` **do**
        if we do not have $sh_{\mathrm{rev},n}$ **then** // Honest closure
            for all received HTLC outputs i of `remoteCom`$_n$ **do**
                if we know the preimage R **then**
                    TX $\leftarrow$ {input: i HTLC output of `remoteCom`$_n$ with $(ph_{\mathrm{htlc},n}, R)$ as method, output: pk_{Alice}}
                    sig $\leftarrow$ `signature` $(TX, sh_{\mathrm{htlc},n})$
                    add (sig, TX) to `toSubmit`
                else
                    add (`channel`, `remoteCom`$_n, h, sh_{\mathrm{htlc},n}$) to `unclaimedReceivedHTLCs`
                end if
            end for
            for all unspent offered HTLC outputs i of `remoteCom`$_n$ **do**
                TX $\leftarrow$ {input: i HTLC output of `remoteCom`$_n$ with $ph_{\mathrm{htlc},n}$ as method, output: pk_{Alice}}
                sig $\leftarrow$ `signature` $(TX, sh_{\mathrm{htlc},n})$
                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
        else // malicious closure
            rev $\leftarrow$ TX {inputs: all `remoteCom`$_n$ outputs, choosing $ph_{\mathrm{rev},n}$ method, output: pk_{Alice}}
            sig $\leftarrow$ `signature` $(\mathrm{rev}, sh_{\mathrm{rev},n})$
            add $(\mathrm{sig}, \mathrm{rev})$ to `toSubmit`
        end if
        move `channel` from `channels` to `closedChannels`
    end for
    send (SUBMIT, `toSubmit`) to $\mathcal{G}_{\mathrm{Ledger}}$

Upon receiving (GETNEW) from *Alice*:
    clear `newChannels`, `closedChannels`, `pendingUpdates` and send them to *Alice*

Fig. 9.

Protocol Π_{LN} - invoice

Upon receiving ($\textsc{pay}$, *Bob*, x, $\overrightarrow{\texttt{path}}$, `receipt`) from $\mathcal{E}$:
    ensure that $\overrightarrow{\texttt{path}}$ consists of valid *pchids*
    ensure that the first $pchid \in \overrightarrow{\texttt{path}}$ has the same *pchid* as in `receipt`
    ensure that `receipt` corresponds to the latest version of an open `channel` $\in$ `channels` in which we have at least x.
    choose unique payment ID *payid* // unique for *Alice* and *Bob*
    add (*Bob*, x, $\overrightarrow{\texttt{path}}$, `receipt`, *payid*, "waiting for invoice") to `pendingPay`
    send ($\textsc{sendInvoice}$, *payid*) to *Bob*

Upon receiving ($\textsc{sendInvoice}$, *payid*) from *Bob*:
    ensure there is no (*Bob*, *payid*) entry in `pendingGetPaid`
    choose random, unique preimage R
    add (*Bob*, R, *payid*) to `pendingGetPaid`
    send ($\textsc{invoice}$, $\mathcal{H}(R)$, *payid*) to *Bob*

Upon receiving ($\textsc{invoice}$, h, *payid*) from *Bob*:
    ensure there is a (*Bob*, x, $\overrightarrow{\texttt{path}}$, `receipt`, *payid*, "waiting for invoice") entry in `pendingPay`
    ensure h is valid (in the range of $\mathcal{H}$)
    remove entry from `pendingPay`
    send ($\textsc{read}$) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign largest block number to t
    $m \leftarrow$ the concatenation of `length` $\overrightarrow{\texttt{path}}$ $(x, \texttt{remoteDelay}_i)$ pairs, where the last `remoteDelay` is $t + 2k + 1 + \texttt{BobDelay}$ and every previous `remoteDelay` is incremented by $3k + \texttt{RHSDelay}$ *Alice* doesn't know these Bob, RHS delays
    $(\mu_0, \delta_0) \leftarrow \texttt{SphinxCreate}\left(m, \text{ public keys of } \overrightarrow{\texttt{path}} \text{ parties}\right)$
    let $\texttt{remoteCom}_n$ the latest signed remote commitment tx
    reduce simple payment output in `remoteCom` by x
    add an additional $(x, ph_{\mathrm{rev},n+1} \vee (ph_{\mathrm{htlc},n+1} \wedge pt_{\mathrm{htlc},n+1},$ on preimage of $h) \vee ph_{\mathrm{htlc},n+1},$ largest `remoteRelayDelay` absolute) output (all with $n+1$ keys) to `remoteCom`, marked with `HTLCNo`
    increment $\texttt{HTLCNo}_{pchid}$ by one and associate $x, h, pchid$ with it
    mark `HTLCNo` as "sender"
    send ($\textsc{updateAddHtlc}$, first *pchid* of $\overrightarrow{\texttt{path}}, \texttt{HTLCNo}_{pchid}, x, h,$ largest `remoteDelay`, (μ_0, δ_0)) to *pchid* channel counterparty

Fig. 10.

Protocol $\mathit{\Pi}_{\mathrm{LN}}$ - $\textsc{updateAddHtlc}$

Upon receiving ($\textsc{updateAddHtlc}$, *pchid*, $\texttt{HTLCNo}, x, h, \texttt{remoteDelay}, M$) from *Bob*:
    ensure *pchid* corresponds to an open `channel` in `channels` where *Bob* has at least x
    ensure $\texttt{HTLCNo} = \texttt{HTLCNo}_{pchid} + 1$
    $(pchid', x', \texttt{remoteDelay}', \delta) \leftarrow \texttt{SphinxPeel}\left(sk_{Alice}, M\right)$
    if $\delta = \texttt{receiver}$ **then**
        ensure $pchid' = \bot, x = x', \texttt{remoteDelay} = \texttt{remoteDelay}' = 2k + 1 + \texttt{delay}$
        increment $\texttt{HTLCNo}_{pchid}$ by one
        mark `HTLCNo` as "receiver"
    else// We are an intermediary
        ensure $x = x', \texttt{remoteDelay} = \texttt{remoteDelay}' + 3k + \texttt{delay}$
        ensure $\texttt{pchid}'$ corresponds to an open channel in `channels` where we have at least x
        increment $\texttt{HTLCNo}_{pchid}$ by one
        mark `HTLCNo` as "intermediary"
    end if
    let $\texttt{remoteCom}_n$ the latest signed remote commitment tx
    reduce delayed payment output in `remoteCom` by x
    add an $(x, ph_{\mathrm{rev},n+1} \vee (ph_{\mathrm{htlc},n+1} \wedge pt_{\mathrm{htlc},n+1}, \texttt{remoteRelayDelay}$ absolute$) \vee ph_{\mathrm{htlc},n+1},$ on preimage of h) htlc output (all with $n+1$ keys) to `remoteCom`, marked with `HTLCNo`
    if $\delta = \texttt{receiver}$ **then**
        retrieve $R : \mathcal{H}\left(R\right) = h$ from `pendingGetPaid`
        add $(\texttt{HTLCNo}, R)$ to $\texttt{pendingFulfills}_{pchid}$
    else if $\delta \neq \texttt{receiver}$ **then** // Send HTLC to next hop
        retrieve *pchid*$'$ data
        let $\texttt{remoteCom}_n$ the latest signed remote commitment tx
        reduce simple payment output in `remoteCom` by x
        add an additional $(x, ph_{\mathrm{rev},n+1} \vee (ph_{\mathrm{htlc},n+1} \wedge pt_{\mathrm{htlc},n+1},$ on preimage of $h) \vee ph_{\mathrm{htlc},n+1}\texttt{remoteRelayDelay}'$ absolute) output (all with $n+1$ keys) to `remoteCom`, marked with `HTLCNo`
        increment `HTLCNo` by 1
        $M' \leftarrow \texttt{SphinxPrepare}\left(M, \delta, sk_{Alice}\right)$
        send ($\textsc{updateAddHtlc}$, *pchid*$'$, `HTLCNo`, $x, h, \texttt{remoteDelay}', M$) to *pchid*$'$ counterparty
    end if

Fig. 11.

Protocol Π_{LN} - $\textsc{updateFulfillHtlc}$

Upon receiving ($\textsc{updateFulfillHtlc}$, *pchid*, HTLC_no, R) from *Bob*:
    ensure HTLC_no $\leq$ `lastRemoteSigned`, HTLC_no $\leq$ `lastLocalSigned`
    ensure HTLC_no is an offered HTLC (`localCom` has h tied to a public key that we own)
    ensure $\mathcal{H}(R) = h$, where h is the hash in the HTLC with number HTLC_no
    add value of HTLC to delayed payment of `remoteCom`
    remove HTLC output with number HTLC_no from `remoteCom`
    if we have a channel `phcid`′ that has a received HTLC with hash h with number `HTLCNo`′ **then** // We are intermediary
        if `HTLCNo`′ $\leq$ `lastRemoteSigned`′ **then** // HTLC committed
            send ($\textsc{read}$) to $\mathcal{G}_{\mathrm{Ledger}}$ and assign reply to Σ_{Alice}
            if latest $\texttt{remoteCom}'_n \in \Sigma_{Alice}$ **then** // counterparty has gone on-chain
                TX $\leftarrow$ {input: `remoteCom`′ HTLC output with number `HTLCNo`′, output: pk_{Alice}}
                sig $\leftarrow$ $\texttt{signature}(\mathrm{TX}, sh_{\mathrm{htlc},n})$
                send ($\textsc{submit}$, (sig, R, TX)) to $\mathcal{G}_{\mathrm{Ledger}}$ // shouldn't be already spent by remote `HTLCTimeout`
            else// counterparty still off-chain
                send ($\textsc{updateFulfillHtlc}$, *pchid*′, `HTLCNo`, R) to counterparty
            end if
        else// we haven't received $\textsc{revokeAndAck}$
            add (`HTLCNo`′, R) to $\texttt{pendingFulfills}_{pchid'}$
        end if
    end if

Fig. 12.

Protocol Π_{LN} - COMMIT

Upon receiving (COMMIT, *pchid*) from $\mathcal{E}$:
  ensure that there is a $\texttt{channel} \in \texttt{channels}$ with ID *pchid*
  retrieve latest remote commitment tx $\texttt{remoteCom}_n$ in $\texttt{channel}$
  ensure $\texttt{remoteCom} \neq \texttt{remoteCom}_n$ // there are uncommitted updates
  ensure $\texttt{channel}$ is not marked as "waiting for REVOKEANDACK"
  $\texttt{remoteCom}_{n+1} \leftarrow \texttt{remoteCom}$
  $\mathrm{ComSig} \leftarrow \texttt{signature}\left(\texttt{remoteCom}_{n+1}, sh_F\right)$
  $\mathrm{HTLCSigs} \leftarrow \emptyset$
  for i from $\texttt{lastRemoteSigned}$ to $\texttt{HTLCNo}$ **do**
    $\texttt{remoteHTLC}_{n+1,i} \leftarrow$ TX {input: HTLC output i of $\texttt{remoteCom}_{n+1}$, output: $(c_{\mathrm{htlc,i}}, ph_{\mathrm{rev},n+1} \vee (pt_{\mathrm{dpay},n+1}, \texttt{delay} + k + 1 \text{ relative}))$}
    add $\texttt{signature}\left(\texttt{remoteHTLC}_{n+1,i}, sh_{\mathrm{htlc},n+1}\right)$ to HTLCSigs
  end for
  add $\texttt{signature}\left(\texttt{remoteHTLC}_{n+1,m+1}, sh_{\mathrm{htlc},n+1}\right)$ to HTLCSigs
  $\texttt{lastRemoteSigned} \leftarrow \texttt{HTLCNo}$
  mark $\texttt{channel}$ as "waiting for REVOKEANDACK"
  send (COMMITMENTSIGNED, *pchid*, ComSig, HTLCSigs) to *pchid* counterparty

Fig. 13.

Protocol Π_{LN} - COMMITMENTSIGNED

Upon receiving (COMMITMENTSIGNED, *pchid*, $\texttt{comSig}_{n+1}$, $\texttt{HTLCSigs}_{n+1}$) from *Bob*:
  ensure that there is a $\texttt{channel} \in \texttt{channels}$ with ID *pchid* with *Bob*
  retrieve latest local commitment tx $\texttt{localCom}_n$ in $\texttt{channel}$
  ensure $\texttt{localCom} \neq \texttt{localCom}_n$ and $\texttt{localCom} \neq \texttt{pendingLocalCom}$ // there are uncommitted updates
  ensure $\texttt{verify}\left(\texttt{localCom}, \texttt{comSig}_{n+1}, pt_F\right) = \texttt{true}$
  for i from $\texttt{lastLocalSigned}$ to $\texttt{HTLCNo}$ **do**
    $\texttt{localHTLC}_{n+1,i} \leftarrow$ TX {input: HTLC output i of $\texttt{localCom}$, output: $(c_{\mathrm{htlc,i}}, ph_{\mathrm{rev},n+1} \vee (pt_{\mathrm{dpay},n+1}, \texttt{remoteDelay} + k + 1 \text{ relative}))$}
    ensure $\texttt{verify}\left(\texttt{localHTLC}_{n+1,i}, \texttt{HTLCSigs}_{n+1,i}, pt_{\mathrm{htlc},n+1}\right) = \texttt{true}$
  end for
  $\texttt{pendingLocalCom} \leftarrow \texttt{localCom}$
  mark $\texttt{pendingLocalCom}$ as "irrevocably committed"
  $\mathrm{prand}_{n+2} \leftarrow \texttt{PRF}\left(\texttt{seed}, n+2\right)$
  $(sh_{\mathrm{com},n+2}, ph_{\mathrm{com},n+2}) \leftarrow \mathrm{KeyShareGen}\left(1^k; \mathrm{prand}_{n+2}\right)$
  send (REVOKEANDACK, *pchid*, $\mathrm{prand}_n, ph_{\mathrm{com},n+2}$) to *Bob*

Fig. 14.

Protocol Π_{LN} - revokeAndAck

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

- ensure there is a `channel` $\in$ `channels` with *Bob* with ID *pchid* marked as "waiting for revokeAndAck"
- ensure $pk\left(st_{\mathrm{com},n}\right) = pt_{\mathrm{com},n}$
- mark `remoteCom`$_{n+1}$ as "irrevocably committed"
- `localCom`$_{n+1}$ $\leftarrow$ `pendingLocalCom`
- unmark `channel`
- $sh_{\mathrm{rev},n} \leftarrow \texttt{CombineKey}\left(shb_{\mathrm{rev}}, phb_{\mathrm{rev}}, st_{\mathrm{com}n}, pt_{\mathrm{com},n}\right)$
- $ph_{\mathrm{rev},n+2} \leftarrow \texttt{CombinePubKey}\left(phb_{\mathrm{rev}}, pt_{\mathrm{com},n+2}\right)$
- $pt_{\mathrm{rev},n+2} \leftarrow \texttt{CombinePubKey}\left(ptb_{\mathrm{rev}}, ph_{\mathrm{com},n+2}\right)$
- $ph_{\mathrm{dpay},n+2} \leftarrow \texttt{PubKeyGen}\left(phb_{\mathrm{dpay}}, ph_{\mathrm{com},n+2}\right)$
- $pt_{\mathrm{dpay},n+2} \leftarrow \texttt{PubKeyGen}\left(ptb_{\mathrm{dpay}}, pt_{\mathrm{com},n+2}\right)$
- $ph_{\mathrm{pay},n+2} \leftarrow \texttt{PubKeyGen}\left(phb_{\mathrm{pay}}, ph_{\mathrm{com},n+2}\right)$
- $pt_{\mathrm{pay},n+2} \leftarrow \texttt{PubKeyGen}\left(ptb_{\mathrm{pay}}, pt_{\mathrm{com},n+2}\right)$
- $ph_{\mathrm{htlc},n+2} \leftarrow \texttt{PubKeyGen}\left(phb_{\mathrm{htlc}}, ph_{\mathrm{com},n+2}\right)$
- $pt_{\mathrm{htlc},n+2} \leftarrow \texttt{PubKeyGen}\left(ptb_{\mathrm{htlc}}, pt_{\mathrm{com},n+2}\right)$

Fig. 15.

Protocol Π_{LN} - push

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

- ensure that there is a `channel` $\in$ `channels` with ID *pchid*
- choose a member (HTLC_no, R) of `pendingFulfills`$_{pchid}$ that is both in an "irrevocably committed" `remoteCom`$_n$ and `localCom`$_n$
- remove (HTLC_no, R) from `pendingFulfills`$_{pchid}$
- send (updateFulfillHtlc, *pchid*, HTLC_no, R) to *pchid* counterparty

Fig. 16.

Protocol Π_{LN} - close

Upon receiving ($\textsc{closeChannel}$, $\texttt{receipt}$) from $\mathcal{E}$:

    ensure $\texttt{receipt}$ corresponds to an open $\texttt{channel} \in \texttt{channels}$

    assign latest $\texttt{channel}$ sequence number to n

    $\text{HTLCs} \leftarrow \emptyset$

    for every HTLC output $\in \texttt{localCom}_n$ with number i **do**

        $\text{sig} \leftarrow \texttt{signature}\left(\texttt{localHTLC}_{n,i}, sh_{\text{htlc},n}\right)$

        add $\left(\text{sig}, \texttt{HTLCSigs}_{n,i}, \texttt{localHTLC}_{n,i}\right)$ to HTLCs

    end for

    $\text{sig} \leftarrow \texttt{signature}\left(\texttt{localCom}_n, sh_F\right)$

    remove $\texttt{channel}$ from $\texttt{channels}$

    send ($\textsc{submit}$, $\left(\text{sig}, \texttt{remoteSig}_n, \texttt{localCom}_n\right)$, HTLCs) to $\mathcal{G}_{\text{Ledger}}$

Fig. 17.

6 Payment Network Functionality

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

Parameters:

– one-way, collision-free hash function $\mathcal{H}$ (for generating transaction IDs)

Interface (messages from $\mathcal{E}$): check

– (REGISTER)
– (SETDELAY, delay)
– (OPENCHANNEL, self, peer, selfCoins)
– (CLOSECHANNEL, receipt)
– (PAY, peer, coins, path, receipt)

Initialisation:

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

Upon receiving $(\text{REGISTER}, \text{delay}, \text{relayDelay})$ from *Alice*:
    $\texttt{delay}\,(Alice) \leftarrow \text{delay}$
    $\texttt{relayDelay}\,(Alice) \leftarrow \text{relayDelay}$
    $\texttt{pendingUpdates}\,(Alice)\,, \texttt{newChannels}\,(Alice) \leftarrow \emptyset$
    $\texttt{negligent}\,(Alice)\,, \texttt{relayNegligent}\,(Alice) \leftarrow \emptyset$
    send (READ) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and assign largest block number to $\texttt{lastPoll}(Alice)$
    send $(\text{REGISTER}, Alice, \text{delay}, \text{relayDelay}, \text{lastPoll})$ to $\mathcal{S}$

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

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

Upon receiving any message except for (REGISTER) from *Alice*:
    ignore message if *Alice* has not registered

Upon receiving (CORRUPTED, *Alice*) from $\mathcal{S}$:
    add *Alice* to $\texttt{corrupted}$

At the end of each activation: **[Orfeas: can this part completely go?]**
    verify onChainbalance() for all parties is consistent with ledger (if not roll back the state and ignore command of activation).

Fig. 18.

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

Upon receiving (openChannel, *Alice*, *Bob*, x, *tid*) from *Alice*:
    ensure *tid* hasn't been used by *Alice* for opening another channel before
    choose unique channel ID *fchid*
    `pendingOpen` (*fchid*) ← (*Alice*, *Bob*, x, *tid*)
    send (openChannel, *Alice*, *Bob*, x, *fchid*, *tid*) to $\mathcal{S}$

Upon receiving (channelAnnounced, *Alice*, $p_{Alice,F}$, $p_{Bob,F}$, *fchid*, *pchid*, *tid*) from $\mathcal{S}$:
    ensure that there is a `pendingOpen`(*fchid*) entry with temporary id *tid*
    add "*Alice* announced", $p_{Alice,F}$, $p_{Bob,F}$, *pchid* to `pendingOpen`(*fchid*)

Upon receiving (checkNew, *Alice*, *Bob*, *tid*) from *Alice*:
    ensure there is a matching `channel` in `pendingOpen`(*fchid*), marked with "*Alice* announced"
    (funder, fundee, x, $p_{Alice,F}$, $p_{Bob,F}$) ← `pendingOpen` (*fchid*)
    send (read) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and assign reply to Σ_{Alice}
    ensure that there is a TX $F \in \Sigma_{Alice}$ with a $(x, (p_{\text{funder},F} \wedge p_{\text{fundee},F}))$ output such that $\mathcal{H}(F) = pchid$
    mark `pendingOpen`(*fchid*) with "*Alice* checked"
    if `pendingOpen`(*fchid*) is not marked with "noted" **then**
        mark `pendingOpen`(*fchid*) with "noted"
        if funder = *Alice* **then**
            offChainBalance (*Alice*) ← offChainBalance (*Alice*) + x **[Orfeas: remove on/offChainBalance?]**
            onChainBalance (*Alice*) ← offChainBalance (*Alice*) − x
            `channel` ← (*Alice*, *Bob*, x, 0, 0, *fchid*, *pchid*)
        else// *Bob* is the funder
            offChainBalance (*Bob*) ← offChainBalance (*Bob*) + x **[Orfeas: remove on/offChainBalance?]**
            onChainBalance (*Bob*) ← offChainBalance (*Bob*) − x
            `channel` ← (*Bob*, *Alice*, x, 0, 0, *fchid*, *pchid*)
        end if
        add `channel` to `channels`
    end if
    add `receipt`(`channel`) to `newChannels`(*Alice*)
    if `pendingOpen`(*fchid*) is marked with "*Alice* checked" and "*Bob* checked" **then**
        clear `pendingOpen`(*fchid*) entry
    end if
    send (channelOpened, *Alice*, *fchid*) to $\mathcal{S}$

Fig. 19.

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

Upon receiving $\left(\textsc{pay}, \mathit{Bob}, x, \overrightarrow{\texttt{path}}, \texttt{receipt}\right)$ from *Alice*:
ensure that $(\mathit{Alice}, c) \in \texttt{receipt}$ and $c \geq x$
ensure that there is a $\texttt{channel} \in \texttt{channels} : \texttt{receipt}\,(\texttt{channel}) = \texttt{receipt}$
ensure that $\overrightarrow{\texttt{path}}$ consists of channels $\in \texttt{channels}$
ensure that each consecutive pair of channels in $\overrightarrow{\texttt{path}}$ has a common member
choose unique payment ID *payid*
add $\left(\mathit{Alice}, \mathit{Bob}, x, \overrightarrow{\texttt{path}}, \mathit{payid}\right)$ to `pendingPay`
send $\left(\textsc{pay}, \mathit{Alice}, \mathit{Bob}, x, \overrightarrow{\texttt{path}}, \texttt{receipt}, \mathit{payid}\right)$ to $\mathcal{S}$

Upon receiving $(\textsc{resolvePay}, \mathit{payid}, \mathit{Charlie})$ from $\mathcal{S}$:
retrieve $\left(\mathit{Alice}, \mathit{Bob}, x, \overrightarrow{\texttt{path}}\right)$ with ID *payid* and remove it from `pendingPay`
if $(\mathit{Charlie} \neq \mathit{Alice}$ and $\mathit{Charlie} \notin \texttt{corrupted})$ or $\mathit{Charlie} \notin \overrightarrow{\texttt{path}}$ **then**
halt
end if
for all $\texttt{channels} \in \overrightarrow{\texttt{path}}$ starting from the one where *Charlie* pays **do**
in the first iteration, *Charlie* is `payer`. In subsequent iterations, `payer` is the unique player that has received but has not given. The other `channel` party is `payee`
if `payer` has x or more in `channel` **then**
update `channel` to the next version and transfer x from `payer` to `payee`
add `receipt`(`channel`) to both parties' `pendingUpdates`
else
revert all updates and remove them from `pendingUpdates`
end if
end for
$\texttt{offChainBalance}\,(\mathit{Charlie}) \leftarrow \texttt{offChainBalance}\,(\mathit{Charlie}) - x$
$\texttt{offChainBalance}\,(\mathit{Bob}) \leftarrow \texttt{offChainBalance}\,(\mathit{Bob}) + x$

Fig. 20.

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

Upon receiving (closeChannel, `receipt`) from *Alice* **[Aggelos: (or $\mathcal{S}$) ??]** **[Orfeas: @Aggelos: why $\mathcal{S}$?]**

  ensure that there is a `channel` $\in$ `channels` : `receipt` (`channel`) = `receipt`

  retrieve *fchid* from `channel`

  `pendingClose` (*fchid*) $\leftarrow$ *Alice*

  send (closeChannel, *fchid*, *Alice*) to $\mathcal{S}$

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

  Alice $\leftarrow$ `pendingClose` (*fchid*)

  retrieve *Charlie*, *Bob*, x, y, *pchid* from `channel` with ID *fchid*

  ensure that *Charlie* = *Alice*

  send (read) to $\mathcal{G}_{\text{Ledger}}$ as *Alice* and assign reply to Σ_{Alice}

  ensure that transaction with ID *pchid* is in Σ_{Alice}, is spent, x of its coins are spendable or will be spendable exclusively by *Alice* and y of its coins are spendable exclusively by *Bob*

  `pendingClose` (*fchid*) $\leftarrow \perp$

  add receipt of `channel` to `closedChannels`(*Bob*)

  remove `channel` from `channels`

  `onChainBalance` (*Alice*) $\leftarrow$ `onChainBalance` (*Alice*) $+ x$

  `onChainBalance` (*Bob*) $\leftarrow$ `onChainBalance` (*Bob*) $+ y$

  `offChainBalance` (*Alice*) $\leftarrow$ `offChainBalance` (*Alice*) $- x$

  `offChainBalance` (*Bob*) $\leftarrow$ `offChainBalance` (*Bob*) $- y$

  send (channelClosed, receipt from `channel`) to *Alice*

Fig. 21.

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

Upon receiving (POLL) from *Alice*:

    `toReport` (*Alice*) $\leftarrow \emptyset$

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

    assign largest block number in Σ_{Alice} to t

    if `lastPoll` (*Alice*) + `delay` (*Alice*) $< t$ **then**

        add [`lastpoll` (*Alice*) , $t -$ `delay` (*Alice*) $- 1$] to `negligent`(*Alice*)

    end if

    if `lastPoll` (*Alice*) + `relayDelay` (*Alice*) $< t$ **then**

        add [`lastpoll` (*Alice*) , $t -$ `relayDelay` (*Alice*) $- 1$] to `relayNegligent`(*Alice*)

    end if

    `lastPoll` (*Alice*) $\leftarrow t$

    scan Σ_{Alice} for honestly closed channels that contain *Alice* and exist in `channels` (txs that spend funding txs that have the same channel version as stored), remove them from `channels` and add them to `toReport`(*Alice*) (marked as "honest")

    scan Σ_{Alice} for maliciouslsy closed channels that contain *Alice* and exist in `channels` (txs that spend funding txs that have an older channel version than stored)

    for all maliciously closed `channel`s of which the spending txs can still be spent by *Alice* **do** // If *Alice* is negligent, she may be unable to punish

        if *Alice* cannot spend those spending txs and *Alice* has not been negligent in the last interval **then**

            halt

        end if

        add `channel` to `toReport`(*Alice*) (marked as "malicious")

    end for

    send (GETCLOSEDFUNDS, `toReport`(*Alice*), *Alice*) to $\mathcal{S}$

Upon receiving (CHANNELSCLOSED, `details`, *Alice*) from $\mathcal{S}$:

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

    for all `channel` $\in$ `details` **do**

        ensure `channel` $\in$ `toReport` (*Alice*)

        if `channel` is marked as "malicious" **then**

            ensure that transactions that spend the funding tx of `channel` and pay *Alice* the entire `channel` value exist in Σ_{Alice}

        else // `channel` is marked as "honest"

            ensure that transactions that spend the funding tx of `channel` and pay *Alice* the her part in `channel` exist in Σ_{Alice}

            ensure that *Alice* has not suffered losses for multi-hop payments where she was not the payer

        end if

        add the receipt of `channel` to `closedChannels`(*Alice*)

        remove `channel` from `channels` and `toReport`(*Alice*)

    end for

Fig. 22.

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

Upon receiving (PUSH, *pchid*) from *Alice*:
  send (PUSH, *pchid*, *Alice*) to $\mathcal{S}$

Upon receiving (COMMIT, *pchid*) from *Alice*:
  send (COMMIT, *pchid*, *Alice*) to $\mathcal{S}$

Upon receiving (GETNEW) from *Alice*:
  clear `newChannels`(*Alice*), `closedChannels`(*Alice*), `pendingUpdates`(*Alice*) and send them to *Alice*

Fig. 23.

7 Security Proof

Functionality $\mathcal{F}_{\text{PayNet,dummy}}$

Upon receiving any message M from *Alice*:
  if M is a valid $\mathcal{F}_{\text{PayNet}}$ message from a player **then**
    send $(M, Alice)$ to $\mathcal{S}$
  end if

Upon receiving any message $(M, Alice)$ from $\mathcal{S}$:
  if M is a valid $\mathcal{F}_{\text{PayNet}}$ message from $\mathcal{S}$ **then**
    send M to *Alice*
  end if

Fig. 24.

Simulator $\mathcal{S}_{\mathrm{LN}}$

Expects the same messages as the protocol, but messages that the protocol expects to receive from $\mathcal{E}$, the simulator expects to receive from $\mathcal{F}_{\mathrm{PayNet,dummy}}$ with the name of the player appended. The simulator internally executes one copy of the protocol per player. Upon receiving any message, the simulator runs the relevant code of the protocol copy tied to the appended player name. Mimicking the real-world case, if a protocol copy sends a message to another player, that message is passed to $\mathcal{A}$ as if sent by the player and if $\mathcal{A}$ allows the message to reach the receiver, then the simulator reacts by acting upon the message with the protocol copy corresponding to the recipient player. A message sent by a protocol copy to $\mathcal{E}$ will be routed by $\mathcal{S}$ to $\mathcal{F}_{\mathrm{PayNet,dummy}}$ instead. To distinguish which player it comes from, $\mathcal{S}$ also appends the player name to the message.

Fig. 25.

Lemma 1. $\textsc{Exec}^{\mathcal{G}_{\mathrm{Ledger}}}_{\Pi_{\mathrm{LN}},\mathcal{A}_{\mathrm{d}},\mathcal{E}} = \textsc{Exec}^{\mathcal{F}_{\mathrm{PayNet,dummy}},\mathcal{G}_{\mathrm{Ledger}}}_{\mathcal{S}_{\mathrm{LN}},\mathcal{E}}$

Proof. Consider a message that $\mathcal{E}$ sends. In the real world, the protocol ITIs produce an output. In the ideal world, the message is given to $\mathcal{S}_{\mathrm{LN}}$ through $\mathcal{F}_{\mathrm{PayNet,dummy}}$. The former simulates the protocol ITIs of the real world (along with their coin flips) and so produces an output from the exact same distribution, which is given to $\mathcal{E}$ through $\mathcal{F}_{\mathrm{PayNet,dummy}}$. Thus the two outputs are indistinguishable. $\square$

Functionality $\mathcal{F}_{\mathrm{PayNet,Reg}}$

For messages REGISTER, REGISTERDONE and REGISTERED, act like $\mathcal{F}_{\mathrm{PayNet}}$.

Upon receiving any other message M from *Alice*:
    if M is a valid $\mathcal{F}_{\mathrm{PayNet}}$ message from a player **then**
        send (M, Alice) to $\mathcal{S}$
    end if

Upon receiving any other message (M, Alice) from $\mathcal{S}$:
    if M is a valid $\mathcal{F}_{\mathrm{PayNet}}$ message from $\mathcal{S}$ **then**
        send M to *Alice*
    end if

Fig. 26.

Simulator $\mathcal{S}_{\mathrm{LN-Reg}}$

Like $\mathcal{S}_{\mathrm{LN}}$, but it does not accept (REGISTERED) from $\mathcal{F}_{\mathrm{PayNet,Reg}}$. Additional differences:

1: Upon receiving (REGISTER, *Alice*, delay, relayDelay, lastPoll) from $\mathcal{F}_{\mathrm{PayNet,Reg}}$:
2: `delay` of *Alice* ITI $\leftarrow$ delay
3: `relayDelay` of *Alice* ITI $\leftarrow$ relayDelay
4: `lastPoll` of *Alice* ITI $\leftarrow$ lastPoll
5: (pk_{Alice}, sk_{Alice}) of *Alice* ITI $\leftarrow$ `KeyGen`()
6: send (REGISTERDONE, *Alice*, pk_{Alice}) to $\mathcal{F}_{\mathrm{PayNet,Reg}}$

Fig. 27.

Lemma 2. $\textsc{Exec}^{\mathcal{F}_{\mathrm{PayNet,dummy}},\mathcal{G}_{\mathrm{Ledger}}}_{\mathcal{S}_{\mathrm{LN}},\mathcal{E}} = \textsc{Exec}^{\mathcal{F}_{\mathrm{PayNet,Reg}},\mathcal{G}_{\mathrm{Ledger}}}_{\mathcal{S}_{\mathrm{LN-Reg}},\mathcal{E}}$

Proof. When $\mathcal{E}$ sends (REGISTER, delay, relayDelay) to *Alice*, it receives as a response (REGISTER, *Alice*, delay, relayDelay, pk_{Alice}) where pk_{Alice} is a public key generated by `KeyGen`() both in the real (c.f. Fig. 1, line 9) and in the ideal world (c.f. Fig. 27, line 5).

Furthermore, one (READ) is sent to $\mathcal{G}_{\mathrm{Ledger}}$ from *Alice* in both cases (Fig. 1, line 8 and Fig. 18, line 8).

Additionally, $\mathcal{S}_{\mathrm{LN-Reg}}$ ensures that the state of *Alice* ITI is exactly the same as what would have been in the case of $\mathcal{S}_{\mathrm{LN}}$, as lines 6-9 of Fig. 1 change the state of *Alice* ITI in the same way as lines 2-5 of Fig. 27.

Lastly, the fact that the state of the *Alice* ITIs are changed in the same way in both worlds, along with the same argument as in the proof of Lemma 1 ensures that the rest of the messages are responded in an indistinguishable way in both worlds. $\square$

Functionality $\mathcal{F}_{\text{PayNet},\text{Open}}$

For messages register, registerDone, registered, openChannel, channelAnnounced and checkNew, act like $\mathcal{F}_{\text{PayNet}}$.

Upon receiving any other message M from *Alice*:
    if M is a valid $\mathcal{F}_{\text{PayNet}}$ message from a player **then**
        send (M, Alice) to $\mathcal{S}$
    end if

Upon receiving any other message (M, Alice) from $\mathcal{S}$:
    if M is a valid $\mathcal{F}_{\text{PayNet}}$ message from $\mathcal{S}$ **then**
        send M to *Alice*
    end if

Fig. 28.

Simulator $\mathcal{S}_{\mathrm{LN-Reg-Open}}$

Like $\mathcal{S}_{\mathrm{LN-Reg}}$. Differences:

Upon receiving (openChannel, *Alice*, *Bob*, x, *fchid*, *tid*) from $\mathcal{F}_{\mathrm{PayNet,Open}}$:
    if both *Alice* and *Bob* are honest **then**
        Simulate the interaction between *Alice* and *Bob* in their respective ITI, as defined in Figures 2-6. All messages should be handed to and received from $\mathcal{A}$, as in the real world execution.
        After sending (fundingSigned) as *Bob* to *Alice*, send (channelAnnounced, $Bob, p_{Alice,F}, p_{Bob,F}, fchid, pchid, tid$) to $\mathcal{F}_{\mathrm{PayNet,Open}}$.
        After submitting F to $\mathcal{G}_{\mathrm{Ledger}}$ as *Alice*, send (channelAnnounced, $Alice, p_{Alice,F}, p_{Bob,F}, fchid, pchid$) to $\mathcal{F}_{\mathrm{PayNet,Open}}$.
    else if *Alice* is honest, *Bob* is corrupted **then**
        Simulate *Alice*'s part of the interaction between *Alice* and *Bob* in *Alice*'s ITI, as defined in Figures 2, 4, and 6.All messages should be handed to and received from $\mathcal{A}$, as in the real world execution.
        After submitting F to $\mathcal{G}_{\mathrm{Ledger}}$ as *Alice*, send (channelAnnounced, $Alice, p_{Alice,F}, p_{Bob,F}, fchid, pchid$) to $\mathcal{F}_{\mathrm{PayNet,Open}}$.
    else if *Alice* is corrupted, *Bob* is honest **then**
        send (openChannel, *Alice*, *Bob*, x, *fchid*, *tid*) to simulated (corrupted) *Alice*
        Simulate *Bob*'s part of the interaction between *Alice* and *Bob* in *Bob*'s ITI, as defined in Figures 3 and 5. All messages should be handed to and received from $\mathcal{A}$, as in the real world execution.
        After sending (fundingSigned) as *Bob* to *Alice*, send (channelAnnounced, $Bob, p_{Alice,F}, p_{Bob,F}, fchid, pchid$) to $\mathcal{F}_{\mathrm{PayNet,Open}}$.
    else if both *Alice* and *Bob* are corrupted **then**
        forward message to $\mathcal{A}$ // $\mathcal{A}$ may open the channel or not
    end if

Upon receiving (channelOpened, *Alice*, *fchid*) from $\mathcal{F}_{\mathrm{PayNet,Open}}$:
    execute lines 6-14 of Fig. 7 with *Alice*'s ITI
    if *Bob* is honest **then**
        expect the delivery of *Alice*'s (fundingLocked) message from $\mathcal{A}$
        simulate Fig. 8 with received message in *Bob*'s ITI
    end if

Upon receiving (checkNew, *Alice*, *Bob*, *tid*) from $\mathcal{F}_{\mathrm{PayNet,Open}}$: // *Alice* should be corrupted
    send (checkNew, *Alice*, *Bob*, *tid*) as $\mathcal{E}$ to $\mathcal{A}$
    if *Bob* is honest **then**
        expect a (fundingLocked) message from $\mathcal{A}$
        simulate Fig. 8 with received message in *Bob*'s ITI
    end if

Fig. 29.

Lemma 3. $\textsc{Exec}^{\mathcal{F}_{\text{PayNet,Reg}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S}_{\text{LN-Reg}},\mathcal{E}} = \textsc{Exec}^{\mathcal{F}_{\text{PayNet,Open}},\mathcal{G}_{\text{Ledger}}}_{\mathcal{S}_{\text{LN-Reg-Open}},\mathcal{E}}$

Proof. When $\mathcal{E}$ sends (openChannel, *Alice*, *Bob*, x, *fchid*, *tid*) to *Alice*, the interaction of Figures 2-6 will be executed in both the real and the ideal world. In more detail, in the ideal world the execution of the honest parties will be simulated by the respective ITIs run by $\mathcal{S}_{\text{LN-Reg-Open}}$, so their state will be identical to that of the parties in the real execution. Furthermore, since $\mathcal{S}_{\text{LN-Reg-Open}}$ executes faithfully the protocol code, it generates the same messages as would be generated by the parties themselves in the real-world setting.

We observe that the input validity check executed by $\mathcal{F}_{\text{PayNet,Open}}$ (Fig. 19, line 2) filters only messages that would be ignored by the real protocol as well and would not change its state either (Fig. 2, line 2).

We also observe that, upon receiving openChannel or channelAnnounced, $\mathcal{F}_{\text{PayNet,Open}}$ does not send any messages to parties other than $\mathcal{S}_{\text{LN-Reg-Open}}$, so we don't have to simulate those.

When $\mathcal{E}$ sends (checkNew, *Alice*, *Bob*, *tid*) to *Alice* in the real world, line 3 of Fig. 7 will allow execution to continue if there exists an entry with temporary id *tid* in `pendingOpen` marked as "broadcast". Such an entry can be added either in Fig. 2, line 7 or in Fig. 3, line 6. The former event can happen only in case *Alice* received a valid openChannel message from *Bob* with temporary id *tid*, which in turn can be triggered only by a valid openChannel message with the same temporary id from $\mathcal{E}$ to *Bob*, whereas the latter only in case *Alice* received a valid openChannel message from $\mathcal{E}$ with the same temporary id. Furthermore, in the first case the "broadcast" mark can be added only before *Alice* sends (fundingSigned, *pchid*, sig) to *Bob* (Fig. 5, line 11) (which needs a valid *Alice*-*Bob* interaction up to that point more in-depth?), and in the second case the "broadcast" mark can be added only before *Alice* sends (submit, (sig, F)) to $\mathcal{G}_{\text{Ledger}}$ (Fig. 6, line 8) (which also needs a valid *Alice*-*Bob* interaction up to that point more in-depth?)

When $\mathcal{E}$ sends (checkNew, *Alice*, *Bob*, *tid*) to *Alice* in the ideal world, line 10 of Fig. 19 will allow execution to continue if there exists an entry with temporary id *tid* and member *Alice* marked as "*Alice* announced" in `pendingOpen`(*fchid*) for some *fchid*. This can only happen if line 8 of Fig. 19 is executed, where `pendingOpen`(*fchid*) contains *tid* as temporary id. This line in turn can only be executed if $\mathcal{F}_{\text{PayNet,Open}}$ received (channelAnnounced, *Alice*, $p_{Alice,F}$, $p_{Bob,F}$, *fchid*, *pchid*, *tid*) from $\mathcal{S}_{\text{LN-Reg-Open}}$ such that `pendingOpen`(*fchid*) exists and has temporary id *tid*, as mandated by line 7 of Fig. 19. Such a message is sent by $\mathcal{S}_{\text{LN-Reg-Open}}$ of Fig. 29 either in lines 5/8, or in lines 4/12. One of the first pair of lines is executed only if $\mathcal{S}_{\mathrm{LN-Reg-Open}}$ receives (openChannel, *Alice*, *Bob*, x, *fchid*, *tid*) from $\mathcal{F}_{\mathrm{PayNet,Open}}$ and the simulated $\mathcal{A}$ allows a valid *Alice*-*Bob* interaction up to the point where *Alice* sends (submit) to $\mathcal{G}_{\mathrm{Ledger}}$, whereas one of the second pair of lines is executed only if $\mathcal{S}_{\mathrm{LN-Reg-Open}}$ receives (openChannel, *Bob*, *Alice*, x, *fchid*, *tid*) from $\mathcal{F}_{\mathrm{PayNet,Open}}$ and the simulated $\mathcal{A}$ allows a valid *Alice*-*Bob* interaction up to the point where *Alice* sends (fundingSigned) to *Bob*.

The last two points lead us to deduce that line 10 of Fig. 19 in the ideal and line 3 of Fig. 7 in the real world will allow execution to continue in the exact same cases with respect to the messages that $\mathcal{E}$ and $\mathcal{A}$ send. Given that execution continues, *Alice* subsequently sends (read) to $\mathcal{G}_{\mathrm{Ledger}}$ and performs identical checks in both the ideal (Fig. 19, lines 12-13) and the real world (Fig. 7, lines 4-5).

Moving on, in the real world lines 6-14 of Fig. 7 are executed by *Alice* and, given that $\mathcal{A}$ allows it, the code of Fig. 8 is executed by *Bob*. Likewise, in the ideal world, the functionality executes lines 14-32 and as a result it (always) sends (channelOpened, *Alice*, *fchid*) to $\mathcal{S}$. In turn $\mathcal{S}$ simulates lines 6-14 of Fig. 7 with *Alice*'s ITI and, if $\mathcal{A}$ allows it, $\mathcal{S}$ simulates the code of Fig. 8 with *Bob*'s ITI. Once more we conclude that both worlds appear to behave identically to both $\mathcal{E}$ and $\mathcal{A}$ under the same inputs from them. □

Functionality $\mathcal{F}_{\mathrm{PayNet,Pay}}$

For messages pay, push and commit, act like $\mathcal{F}_{\mathrm{PayNet}}$.

Upon receiving any other message M from *Alice*:
    if M is a valid $\mathcal{F}_{\mathrm{PayNet}}$ message from a player **then**
        send $(M, Alice)$ to $\mathcal{S}$
    end if

Upon receiving any other message $(M, Alice)$ from $\mathcal{S}$:
    if M is a valid $\mathcal{F}_{\mathrm{PayNet}}$ message from $\mathcal{S}$ **then**
        send M to *Alice*
    end if

Fig. 30.

Simulator $\mathcal{S}_{\mathrm{LN-Reg-Open-Pay}}$

Like $\mathcal{S}_{\mathrm{LN-Reg-Open}}$. Differences:

1: Upon receiving $\left(\textsc{pay}, \mathit{Alice}, \mathit{Bob}, x, \overrightarrow{\texttt{path}}, \texttt{receipt}, \mathit{payid}\right)$ from $\mathcal{S}_{\mathrm{LN-Reg-Open-Pay}}$:

2: Simulate payment from *Alice* to *Bob* along $\overrightarrow{\texttt{path}}$ continue

3: send $(\textsc{resolvePay}, \mathit{payid}, \mathit{Charlie})$ to $\mathcal{S}_{\mathrm{LN-Reg-Open-Pay}}$

Fig. 31.

Lemma 4. $\textsc{Exec}^{\mathcal{F}_{\mathrm{PayNet,Open}}, \mathcal{G}_{\mathrm{Ledger}}}_{\mathcal{S}_{\mathrm{LN-Reg-Open}}, \mathcal{E}} = \textsc{Exec}^{\mathcal{F}_{\mathrm{PayNet,Pay}}, \mathcal{G}_{\mathrm{Ledger}}}_{\mathcal{S}_{\mathrm{LN-Reg-Open-Pay}}, \mathcal{E}}$

Proof. □

8 Combined Sign primitive

8.1 Algorithms

- $(\mathit{mpk}, \mathit{msk}) \leftarrow \textsc{MasterKeyGen}\left(1^k\right)$
- $(\mathit{pk}, \mathit{sk}) \leftarrow \textsc{KeyShareGen}\left(1^k\right)$
- $(\mathit{cpk}_l, \mathit{csk}_l) \leftarrow \textsc{CombineKey}\left(\mathit{msk}, \mathit{mpk}, \mathit{sk}, \mathit{pk}\right)$
- $\mathit{cpk}_l \leftarrow \textsc{CombinePubKey}\left(\mathit{mpk}, \mathit{pk}\right)$
- $\sigma \leftarrow \textsc{Sign}\left(\mathit{csk}, m\right)$
- $\{0, 1\} \leftarrow \textsc{Verify}\left(\mathit{cpk}, m, \sigma\right)$

8.2 Correctness

- $\forall k \in \mathcal{N}$,
  $\Pr[(\mathit{mpk}, \mathit{msk}) \leftarrow \textsc{MasterKeyGen}\left(1^k\right)$,
  $(\mathit{pk}, \mathit{sk}) \leftarrow \textsc{KeyShareGen}\left(1^k\right)$,
  $(\mathit{cpk}_1, \mathit{csk}_1) \leftarrow \textsc{CombineKey}\left(\mathit{msk}, \mathit{mpk}, \mathit{sk}, \mathit{pk}\right)$,
  $\mathit{cpk}_2 \leftarrow \textsc{CombinePubKey}\left(\mathit{mpk}, \mathit{pk}\right)$,
  $\mathit{cpk}_1 = \mathit{cpk}_2] = 1$
- $\forall k \in \mathcal{N}, m \in \mathcal{M}$,
  $\Pr[(\mathit{mpk}, \mathit{msk}) \leftarrow \textsc{MasterKeyGen}\left(1^k\right)$,
  $(\mathit{pk}, \mathit{sk}) \leftarrow \textsc{KeyShareGen}\left(1^k\right)$,
  $(\mathit{cpk}, \mathit{csk}) \leftarrow \textsc{CombineKey}\left(\mathit{mpk}, \mathit{msk}, \mathit{pk}, \mathit{sk}\right)$,
  $\textsc{Verify}(\mathit{cpk}, m, \textsc{Sign}(\mathit{csk}, m)) = 1] = 1$

8.3 Security

Game share-EUF$^{\mathcal{A}}\left(1^k\right)$

$(\mathtt{aux}, mpk, n) \leftarrow \mathcal{A}(\textsc{init})$
for $i \leftarrow 1$ to n **do**
  $(pk_i, sk_i) \leftarrow \textsc{KeyShareGen}\left(1^k\right)$
end for
$(cpk^*, pk^*, m^*, \sigma^*) \leftarrow \mathcal{A}(\textsc{keys}, \mathtt{aux}, pk_1, \ldots, pk_n)$
if $pk^* \in \{pk_1, \ldots, pk_n\} \wedge cpk^* = \textsc{CombinePubKey}(mpk, pk^*) \wedge \textsc{Verify}(cpk^*, m^*, \sigma^*) = 1$ **then**
  return 1
else
  return 0
end if

Fig. 32.

Definition 1. *A Combined Sign scheme is* share-EUF-secure *if*

$$\forall k \in \mathbb{N}, \forall \mathcal{A} \in \mathsf{PPT}, \Pr\left[\text{share-EUF}^{\mathcal{A}}\left(1^k\right) = 1\right] < negl(k)$$

Game master-EUF-CMA$^{\mathcal{A}}\left(1^k\right)$

$(mpk, msk) \leftarrow \textsc{MasterKeyGen}\left(1^k\right)$
$i \leftarrow 0$
$(\mathtt{aux}_i, \text{response}) \leftarrow \mathcal{A}(\textsc{init}, mpk)$
while response can be parsed as (pk, sk, m) **do**
  $i \leftarrow i + 1$
  store pk, sk, m as pk_i, sk_i, m_i
  $(cpk_i, csk_i) \leftarrow \textsc{CombineKey}(mpk, msk, pk_i, sk_i)$
  $\sigma_i \leftarrow \textsc{Sign}(csk_i, m_i)$
  $(\mathtt{aux}_i, \text{response}) \leftarrow \mathcal{A}(\textsc{signature}, \mathtt{aux}_{i-1}, \sigma_i)$
end while
parse response as $(cpk^*, pk^*, m^*, \sigma^*)$
if $m^* \notin \{m_1, \ldots, m_i\} \wedge cpk^* = \textsc{CombinePubKey}(mpk, pk^*) \wedge \textsc{Verify}(cpk^*, m^*, \sigma^*) = 1$ **then**
  return 1
else
  return 0
end if

Fig. 33.

Definition 2. *A Combined Sign scheme is* master-EUF-CMA-secure *if*

$$\forall k \in \mathbb{N}, \forall \mathcal{A} \in \mathsf{PPT}, \Pr\left[\mathsf{master\text{-}EUF\text{-}CMA}^{\mathcal{A}}\left(1^k\right) = 1\right] < negl\left(k\right)$$

Definition 3. *A Combined Sign scheme is* combine-EUF-secure *if it is both share-EUF-secure and master-EUF-CMA-secure.*

8.4 Construction

output standard signing keypairs to avoid duplication?

Parameters: $\mathcal{H}, G$

function $\textsc{MasterKeyGen}(1^k, \text{rand})$
  Return (rand, $G \cdot$ rand)
end function
function $\textsc{KeyShareGen}(1^k, \text{rand})$
  Return (rand, $G \cdot$ rand)
end function
function $\textsc{CombineKey}(msk, mpk, sk, pk)$
  return $msk \cdot \mathcal{H}\left(mpk \parallel pk\right) + sk \cdot \mathcal{H}\left(pk \parallel mpk\right)$
end function
function $\textsc{CombinePubKey}(mpk, pk)$
  return $mpk \cdot \mathcal{H}\left(mpk \parallel pk\right) + pk \cdot \mathcal{H}\left(pk \parallel mpk\right)$
end function
function $\textsc{Sign}(csk, m)$
  like standard sign
end function
function $\textsc{Verify}(cpk, m, \sigma)$
  like standard verify
end function

just to remember
$sh_{\text{rev},n} \leftarrow shb_{\text{rev}} \cdot \mathcal{H}\left(phb_{\text{rev}}||pt_{\text{com},n}\right) + st_{\text{com},n} \cdot \mathcal{H}\left(pt_{\text{com},n}||phb_{\text{rev}}\right)$
$pt_{\text{rev},n+2} \leftarrow ptb_{\text{rev}} \cdot \mathcal{H}\left(ptb_{\text{rev}}||ph_{\text{com},n+2}\right) + ph_{\text{com},n+2} \cdot \mathcal{H}\left(ph_{\text{com},n+2}||ptb_{\text{rev}}\right)$
$ph_{\text{rev},n+2} \leftarrow phb_{\text{rev}} \cdot \mathcal{H}\left(phb_{\text{rev}}||pt_{\text{com},n+2}\right) + pt_{\text{com},n+2} \cdot \mathcal{H}\left(pt_{\text{com},n+2}||phb_{\text{rev}}\right)$

Lemma 5. *The construction above is share-EUF-secure in the Random Oracle model under the assumption that the underlying signature scheme is strongly EUF-CMA-secure and the range of the Random Oracle coincides with that of the underlying signature scheme signing keys.*

Proof. Let $k \in \mathbb{N}, \mathcal{B}$ PPT algorithm such that

$$\Pr\left[\mathsf{share\text{-}EUF}^{\mathcal{B}}\left(1^k\right) = 1\right] = a > \text{negl}\left(k\right) \quad .$$

We construct a PPT distinguisher $\mathcal{A}$ (Fig. 34) such that

$$\Pr\left[\mathsf{EUF\text{-}CMA}^{\mathcal{A}}\left(1^k\right) = 1\right] > \mathrm{negl}\left(k\right)$$

that breaks the assumption, thus proving Lemma 5.

Algorithm $\mathcal{A}\left(vk\right)$

$j \xleftarrow{\$} U\left[1, T\left(\mathcal{B}\right)\right]$ // $T\left(M\right)$ is the maximum running time of M
  Random Oracle: for every first-seen query q from $\mathcal{B}$ set $\mathcal{H}\left(q\right)$ to a random value
  return $\mathcal{H}\left(q\right)$ to $\mathcal{B}$
$\left(\mathsf{aux}, mpk, n\right) \leftarrow \mathcal{A}\left(\textsc{init}\right)$
for $i \leftarrow 1$ to n **do**
  $\left(pk_i, sk_i\right) \leftarrow \textsc{KeyShareGen}\left(1^k\right)$
end for
  Random Oracle: Let q be the jth first-seen query from $\mathcal{B}$:
  if $q = \left(mpk \,\|\, x\right)$ **then**
    if $\mathcal{H}\left(x \,\|\, mpk\right)$ unset **then**
      set $\mathcal{H}\left(x \,\|\, mpk\right)$ to a random value
    end if
    set $\mathcal{H}\left(mpk \,\|\, x\right)$ to $\left(vk - x \cdot \mathcal{H}\left(x \,\|\, mpk\right)\right) \cdot mpk^{-1}$
  else if $q = \left(x \,\|\, mpk\right)$ **then**
    if $\mathcal{H}\left(mpk \,\|\, x\right)$ unset **then**
      set $\mathcal{H}\left(mpk \,\|\, x\right)$ to a random value
    end if
    set $\mathcal{H}\left(x \,\|\, mpk\right)$ to $\left(vk - mpk \cdot \mathcal{H}\left(mpk \,\|\, x\right)\right) \cdot x^{-1}$
  else
    set $\mathcal{H}\left(q\right)$ to a random value
  end if
  return $\mathcal{H}\left(q\right)$ to $\mathcal{B}$
$\left(cpk^*, pk^*, m^*, \sigma^*\right) \leftarrow \mathcal{B}\left(\textsc{keys}, \mathsf{aux}, pk_1, \ldots, pk_n\right)$
if $vk = cpk^* \wedge \mathcal{B}$ wins the $\mathsf{share\text{-}EUF}$ game **then** // $\mathcal{A}$ won the $\mathsf{EUF\text{-}CMA}$ game
  return $\left(m^*, \sigma^*\right)$
else
  return $\textsc{fail}$
end if

Fig. 34.

Let Y be the range of the random oracle. The modified random oracle used in Fig. 34 is indistinguishable from the standard random oracle by PPT algorithms since the statistical distance of the standard random oracle from the modified one is at most $\frac{1}{2|Y|} < negl\left(k\right)$ as they differ in at most one element.

Let E denote the event in which $\mathcal{B}$ does not invoke CombinePubKey to produce cpk^*. In that case the values $\mathcal{H}(pk^* \,\|\, mpk)$ and $\mathcal{H}(mpk \,\|\, pk^*)$ are decided after $\mathcal{B}$ terminates (Fig. 34, line 24) and thus

$$\begin{aligned}\Pr\left[cpk^* = \textsc{CombinePubKey}\left(mpk, pk^*\right) | E\right] = \frac{1}{|Y|} < negl\left(k\right) \Rightarrow \\ \Pr\left[cpk^* = \textsc{CombinePubKey}\left(mpk, pk^*\right) \wedge E\right] < negl\left(k\right) \quad .\end{aligned} \tag{1}$$

It is

$$\begin{gathered}(\mathcal{B} \text{ wins}) \rightarrow \left(cpk^* = \textsc{CombinePubKey}\left(mpk, pk^*\right)\right) \Rightarrow \\ \Pr\left[\mathcal{B} \text{ wins}\right] \leq \Pr\left[cpk^* = \textsc{CombinePubKey}\left(mpk, pk^*\right)\right] \Rightarrow \\ \Pr\left[\mathcal{B} \text{ wins} \wedge E\right] \leq \Pr\left[cpk^* = \textsc{CombinePubKey}\left(mpk, pk^*\right) \wedge E\right] \overset{(1)}{\Rightarrow} \\ \Pr\left[\mathcal{B} \text{ wins} \wedge E\right] < negl\left(k\right) \quad .\end{gathered}$$

But we know that $\Pr\left[\mathcal{B} \text{ wins}\right] = \Pr\left[\mathcal{B} \text{ wins} \wedge E\right] + \Pr\left[\mathcal{B} \text{ wins} \wedge \neg E\right]$ and $\Pr\left[\mathcal{B} \text{ wins}\right] = a$ by the assumption, thus

$$\Pr\left[\mathcal{B} \text{ wins} \wedge \neg E\right] > a - negl\left(k\right) \quad . \tag{2}$$

We now focus at the event $\neg E$. Let F the event in which the call of $\mathcal{B}$ to CombinePubKey to produce cpk^* results in the jth invocation of the Random Oracle. Since j is chosen uniformly at random, $\Pr\left[F|\neg E\right] = \frac{1}{T(\mathcal{B})}$. Observe that $\Pr\left[F|E\right] = 0 \Rightarrow \Pr\left[F\right] = \Pr\left[F|\neg E\right] = \frac{1}{T(\mathcal{B})}$.

In the case where the event $(F \wedge \mathcal{B} \text{ wins} \wedge \neg E)$ holds, it is

$$\begin{gathered}cpk^* = \textsc{CombinePubKey}\left(mpk, pk^*\right) = \\ mpk \cdot \mathcal{H}\left(mpk \,\|\, pk^*\right) + pk^* \cdot \mathcal{H}\left(pk^* \,\|\, mpk\right)\end{gathered}$$

Since F holds, the jth invocation of the Random Oracle queried either $\mathcal{H}(mpk \,\|\, pk^*)$ or $\mathcal{H}(pk^* \,\|\, mpk)$. In either case (Fig. 34, lines 9-18), it is $cpk^* = vk$. This means that $\textsc{Verify}\left(vk, m^*, \sigma^*\right) = 1$. We conclude that in the event $(F \wedge \mathcal{B} \text{ wins} \wedge \neg E)$, $\mathcal{A}$ wins the EUF-CMA game. A final observation is that the probability that the events $(\mathcal{B} \text{ wins} \wedge \neg E)$ and F are almost independent, thus

$$\begin{gathered}\Pr\left[F \wedge \mathcal{B} \text{ wins} \wedge \neg E\right] = \Pr\left[F\right] \Pr\left[\mathcal{B} \text{ wins} \wedge \neg E\right] \pm negl\left(k\right) \overset{(2)}{=} \\ \frac{a - negl\left(k\right)}{T\left(\mathcal{B}\right)} \pm negl\left(k\right) > negl\left(k\right)\end{gathered}$$

$\square$

Lemma 6. *The construction above is* master-EUF-CMA*-secure in the Random Oracle model under the assumption that the underlying signature scheme is strongly* EUF-CMA*-secure and the range of the Random Oracle coincides with that of the underlying signature scheme signing keys.*

Proof. Let $k \in \mathbb{N}, \mathcal{B}$ PPT algorithm such that

$$\Pr\left[\mathsf{master\text{-}EUF\text{-}CMA}^{\mathcal{B}}\left(1^k\right) = 1\right] = a > \mathrm{negl}\left(k\right) \quad .$$

We construct a PPT distinguisher $\mathcal{A}$ (Fig. 35) such that

$$\Pr\left[\mathsf{EUF\text{-}CMA}^{\mathcal{A}}\left(1^k\right) = 1\right] > \mathrm{negl}\left(k\right)$$

that breaks the assumption, thus proving Lemma 6.

Algorithm $\mathcal{A}(vk)$

$j \xleftarrow{\$} U[1, T(\mathcal{B}) + T(\mathcal{A})]$ // $T(M)$ is the maximum running time of M
    Random Oracle: for every first-seen query q from $\mathcal{B}$ set $\mathcal{H}(q)$ to a random value
    return $\mathcal{H}(q)$ to $\mathcal{B}$
$(mpk, msk) \leftarrow \textsc{MasterKeyGen}\left(1^k\right)$
    Random Oracle: Let q be the jth first-seen query from $\mathcal{B}$ or $\mathcal{A}$:
    if $q = (mpk \,\|\, x)$ **then**
        if $\mathcal{H}(x \,\|\, mpk)$ unset **then**
            set $\mathcal{H}(x \,\|\, mpk)$ to a random value
        end if
        set $\mathcal{H}(mpk \,\|\, x)$ to $(vk - x \cdot \mathcal{H}(x \,\|\, mpk)) \cdot mpk^{-1}$
    else if $q = (x \,\|\, mpk)$ **then**
        if $\mathcal{H}(mpk \,\|\, x)$ unset **then**
            set $\mathcal{H}(mpk \,\|\, x)$ to a random value
        end if
        set $\mathcal{H}(x \,\|\, mpk)$ to $(vk - mpk \cdot \mathcal{H}(mpk \,\|\, x)) \cdot x^{-1}$
    else
        set $\mathcal{H}(q)$ to a random value
    end if
    return $\mathcal{H}(q)$ to $\mathcal{B}$ or $\mathcal{A}$
$i \leftarrow 0$
$(\mathtt{aux}_i, \text{response}) \leftarrow \mathcal{B}(\textsc{init}, mpk)$
while response can be parsed as (pk, sk, m) **do**
    $i \leftarrow i + 1$
    store pk, sk, m as pk_i, sk_i, m_i
    $(cpk_i, csk_i) \leftarrow \textsc{CombineKey}(mpk, msk, pk_i, sk_i)$
    $\sigma_i \leftarrow \textsc{Sign}(csk_i, m_i)$
    $(\mathtt{aux}_i, \text{response}) \leftarrow \mathcal{B}(\textsc{signature}, \mathtt{aux}_{i-1}, \sigma_i)$
end while
parse response as $(cpk^*, pk^*, m^*, \sigma^*)$
$(cpk^*, pk^*, m^*, \sigma^*) \leftarrow \mathcal{B}(\textsc{keys}, \mathtt{aux}, pk_1, \ldots, pk_n)$
if $vk = cpk^* \wedge \mathcal{B}$ wins the master-EUF-CMA game **then** // $\mathcal{A}$ won the EUF-CMA game
    return (m^*, σ^*)
else
    return FAIL
end if

Fig. 35.

The modified random oracle used in Fig. 35 is indistinguishable from the standard random oracle for the same reasons as in the proof of Lemma 5.

Let E denote the event in which CombinePubKey is not invoked to produce cpk^*. In that case the values $\mathcal{H}(pk^* \| mpk)$ and $\mathcal{H}(mpk \| pk^*)$ are decided after $\mathcal{B}$ terminates (Fig. 35, line 31) and thus

$$\begin{aligned} &\Pr[cpk^* = \textsc{CombinePubKey}(mpk, pk^*) | E] < negl(k) \Rightarrow \\ &\Pr[cpk^* = \textsc{CombinePubKey}(mpk, pk^*) \wedge E] < negl(k) \quad . \end{aligned} \tag{3}$$

We can reason like in the proof of Lemma 5 to deduce that

$$\Pr[\mathcal{B} \text{ wins} \wedge \neg E] > a - negl(k) \quad . \tag{4}$$

We now focus at the event $\neg E$. Let F the event in which the call of to CombinePubKey that produces cpk^* results in the jth invocation of the Random Oracle. Since j is chosen uniformly at random, $\Pr[F|\neg E] = \frac{1}{T(\mathcal{B})+T(\mathcal{A})}$. Observe that $\Pr[F|E] = 0 \Rightarrow \Pr[F] = \Pr[F|\neg E] = \frac{1}{T(\mathcal{B})+T(\mathcal{A})}$.

Once more we can reason in the same fashion as in the proof of Lemma 5 to deduce that

$$\begin{aligned} &\Pr[F \wedge \mathcal{B} \text{ wins} \wedge \neg E] = \Pr[F] \Pr[\mathcal{B} \text{ wins} \wedge \neg E] \pm negl(k) \overset{(4)}{=} \\ &\frac{a - negl(k)}{T(\mathcal{B}) + T(\mathcal{A})} \pm negl(k) > negl(k) \end{aligned}$$

□

Theorem 1. *The construction above is combine-EUF-secure in the Random Oracle model under the assumption that the underlying signature scheme is strongly EUF-CMA-secure.*

Proof. The construction is combine-EUF-secure as a direct consequence of Lemma 5, Lemma 6 and the definition of combine-EUF-security. □

9 Notes on Lightning Specification

– The relevant part of the specification can be found at `https://github.com/lightningnetwork/lightning-rfc/blob/master/02-peer-protocol.md`.

References

1. Poon J., Dryja T.: The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments