Virtual payment channels for the Lightning Network

Aggelos Kiayias[1,2] and Orfeas Stefanos Thyfronitis Litos[1]

[1] University of Edinburgh
[2] IOHK
akiayias@inf.ed.ac.uk, o.thyfronitis@ed.ac.uk

Abstract. Virtual Lightning-like payment channels

1 Notation

We introduce the following notation to formally express Bitcoin transactions.

Basic building blocks

- signature (needed to spend): $\text{player}_{\text{sigName}}$, e.g. $Alice_{\text{rev}}$
- value (in bitcoin): xɃ

Spending method – in transaction output (possibly named)

- n out of n multisig:
  $\texttt{AND}(\text{sig}_1, \dots, \text{sig}_n)$, alternatively $\text{sig}_1 \wedge \dots \wedge \text{sig}_n$
- relative delay – minimum blocks between current and spending transaction:
  $\texttt{rltvDelay}(n\text{-of-}n\texttt{-multisig}, blocks)$, e.g. $\texttt{rltvDelay}(Alice_F \wedge Bob_F, 3)$
- absolute delay – minimum block where current transaction can be spent:
  $\texttt{absDelay}(n\text{-of-}n\texttt{-multisig}, block)$, e.g. $\texttt{delayed} = \texttt{absDelay}(Alice_{\text{htlc}}, 1005)$
- hashlock – a hash is provided here, its preimage must be provided by the spending transaction. Can be nested: TODO remove nesting if unneeded
  $\texttt{hashLock}(n\text{-of-}n\texttt{-multisig}, h)$, e.g. $\texttt{hashLock}(Alice_{\text{htlc}} \wedge Charlie_{\text{htlc}}, \texttt{0x9b4f})$

Spending methods set – each output contains one such set
$\texttt{OR}(method_1, \dots, method_m)$, alternatively $method_1 \vee \dots \vee method_m$,
e.g. $(\texttt{fulfill} = \texttt{hashLock}(Alice, \texttt{0x1bc6})) \vee (\texttt{refund} = \texttt{absDelay}(Bob, 1007))$

Output – each transaction contains one or more (possibly named)
$\texttt{txOut}(set\ of\ methods, value)$,
e.g. $\texttt{coins}_{Alice} = \texttt{txOut}(\texttt{normal} = \texttt{rltvDelay}(Alice, 10) \vee \texttt{revocation} = Bob_{\text{rev}})$

Input – each transaction contains one or more, unambiguous arguments can be omitted
$\texttt{txIn}(method\ name, list\ of\ signatures, preimage)$ TODO or list of preimages if needed
e.g. $\texttt{txIn}(\texttt{comm}_{Alice}, \texttt{coins}_{Alice}, \texttt{revocation})$

Transaction
$\mathtt{tx}((\mathtt{txIn}_1, \ldots, \mathtt{txIn}_n), (\mathtt{txOut}_1, \ldots, \mathtt{txOut}_m))$,
e.g. $\mathtt{rev}_{Bob} = \mathtt{tx}((\mathtt{txIn}(\mathtt{comm}_{Alice}, \mathtt{coins}_{Alice}, \mathtt{revocation})), (\mathtt{txOut}(Bob)))$

2 Virtual channel over 2 normal channels

We use the term "base channels" for the channels described in [?]. We adopt the notation of Perun [?] to differentiate base and virtual channels: $Peggy \Leftrightarrow Quinney$ refers to a base channel and $Peggy \leftrightarrow Quinney$ refers to a virtual channel.

Fig. TODO reffig:base shows the transactions that two parties hold in an existing base channel between them, as described in [?][3]. Let existing $Alice \Leftrightarrow Bob$, $Bob \Leftrightarrow Charlie$ base channels, with an $Alice \leftrightarrow Charlie$ virtual channel on top. Fig. TODO reffig:virt shows the transactions that the three parties hold.

- On-chain outputs:
  - $\mathtt{funding}_{\mathrm{AB}} = \mathtt{txOut}(Alice_{F,1} \wedge Bob_{F,1}, ab)$
  - $\mathtt{funding}_{\mathrm{BC}} = \mathtt{txOut}(Bob_{F,2} \wedge Charlie_{F,2}, bc)$
- Txs held by *Alice*:
  - $\mathtt{baseCommitment}_{Alice} = \mathtt{tx}($
    $(\mathtt{txIn}(\mathtt{funding}_{\mathrm{AB}}))$,
    $(\mathtt{base} = \mathtt{txOut}(Alice_{F',1} \wedge Bob_{F',1}, ab - ac)$,
    $\mathtt{virt} = \mathtt{txOut}(Alice_{F',2} \wedge Charlie_{F',2}, ac)$,
    $\mathtt{mark} = \mathtt{txOut}(Alice_{M,1} \wedge Bob_{M,1} \wedge Charlie_{M,1}, 0)))$
  - $\mathtt{ABcommitment}_{Alice} = \mathtt{tx}($
    $(\mathtt{txIn}(\mathtt{baseCommitment}_{Alice}, \mathtt{base}))$, TODO common keys make this incomplete
    $(\mathtt{txOut}(\mathtt{rltvDelay}(Alice_{\mathrm{dcom},1}, \mathtt{bobDelay}) \vee Bob_{\mathrm{rev},1}, a - ac)$,
    $(\mathtt{txOut}(Bob_{\mathrm{com},1}, b))))$
  - $\mathtt{ACcommitment}_{Alice} = \mathtt{tx}($
    $(\mathtt{txIn}(\mathtt{baseCommitment}_{Alice}, \mathtt{virt}))$,
    $(\mathtt{txOut}(\mathtt{rltvDelay}(Alice_{\mathrm{dcom},1}, \mathtt{bobDelay}) \vee Bob_{\mathrm{rev},1}, a - ac)$,
    $(\mathtt{txOut}(Bob_{\mathrm{com},1}, b))))$
- Txs held by *Bob*:
  -
  -
  -
- Txs held by *Charlie*:
  -

[3] In the lightning spec (https://github.com/lightningnetwork/lightning-rfc/) really

Functionality $\mathcal{F}_{\text{chan}}^{Alice,Bob,\texttt{delay}}$

if $\texttt{open} = \texttt{false}$ **then**
  ignore any message $\in \{$getBalance, pay, close$\}$
else
  ignore any message $\in \{$open, getSpendingKeys$\}$
end if

Upon receiving (open, keys, coins) from *Alice*:
  $\texttt{coins}(Alice) \leftarrow \text{coins}$
  $\texttt{coins}(Bob) \leftarrow 0$
  $\texttt{open} \leftarrow \texttt{true}$
  $\texttt{channelData} \leftarrow \texttt{openChannel}(Alice, Bob, keys, coins)$ // May run a protocol and/or interact with $\mathcal{G}_{\text{Ledger}}$

Upon receiving (pay, coins) from $Charlie \in \{Alice, Bob\}$:
  if $\texttt{coins}(Charlie) \geq \text{coins} \wedge \texttt{checkOpen}(\texttt{channelData}) = 1$ **then**
    $\texttt{coins}(Charlie) \leftarrow \texttt{coins}(Charlie) - \text{coins}$
    $\texttt{coins}(\text{counterparty}) \leftarrow \texttt{coins}(\text{counterparty}) + \text{coins}$
    return `paid`
  else
    return `notPaid`
  end if

Upon receiving (close) from $Charlie \in \{Alice, Bob\}$:
  $(sk_{Alice}, sk_{Bob}) \leftarrow \texttt{closeChannel}(Alice, Bob, \texttt{channelData})$ // May run a protocol and/or interact with $\mathcal{G}_{\text{Ledger}}$
  $\texttt{coins}(Alice), \texttt{coins}(Alice) \leftarrow 0$
  $\texttt{open} \leftarrow \texttt{false}$
  Send (clock-read, $\mathcal{F}_{\text{chan}}^{Alice,Bob,\texttt{delay}}$) to $\mathcal{G}_{\text{clock}}$ and assign reply to τ_{close}

Upon receiving (getBalance) from $Charlie \in \{Alice, Bob\}$:
  return $(\texttt{coins}(Charlie), \texttt{coins}(\text{counterparty}))$

Upon receiving (getSpendingKeys) from $Charlie \in \{Alice, Bob\}$:
  Send (clock-read, $\mathcal{F}_{\text{chan}}^{Alice,Bob,\texttt{delay}}$) to $\mathcal{G}_{\text{clock}}$ and assign reply to τ
  if $\tau \geq \tau_{\text{close}} + \texttt{delay}$ **then**
    return $sk_{Charlie}$
  end if

Fig. 1.

Definition 1 **($\mathcal{F}_{\text{chan}}^{Alice,Bob,\texttt{delay}}$ Funds retrievability).**
Let Alice honest, Bob corrupted, $\mathcal{F}_{\text{chan}}^{Alice,Bob,\texttt{delay}}$ parametrized by three protocols `openChannel`, `closeChannel`, and `checkOpen`, interacting with $\mathcal{G}_{\text{Ledger}}$ which in

turn is parametrized by the `validate` *predicate of Definition* TODO *. We say that this* $\mathcal{F}_{\text{chan}}^{Alice,Bob,\texttt{delay}}$ *provides* funds retrievability *if for any PPT* $\mathcal{A}$*, after* $\mathcal{F}_{\text{chan}}^{Alice,Bob,\texttt{delay}}$ *serves a sequence of*

$$\begin{gathered}((\textsc{open}, \text{keys}, \text{coins}_0, Alice), \texttt{paid} \leftarrow (\textsc{pay}, \text{coins}_1^{Alice}, Alice),\\ \texttt{paid} \leftarrow (\textsc{pay}, \text{coins}_1^{Bob}, Bob), \dots, \texttt{paid} \leftarrow (\textsc{pay}, \text{coins}_m^{Bob}, Bob),\\ \texttt{paid} \leftarrow (\textsc{pay}, \text{coins}_n^{Alice}, Alice), (\textsc{close}, Charlie \in \{Alice, Bob\}),\\ sk_{Alice} \leftarrow (\textsc{getSpendingKeys}, Alice))\end{gathered}$$

requests (possibly interspersed with any number of invalid requests by entities other than Alice) and after the response sk_{Alice} *from the* ($\textsc{getSpendingKeys}$) *message is received, it can be used at any point in time to sign a valid transaction for* $\mathcal{G}_{\text{Ledger}}$ *that spends an existing, unspent output of value*

$$\text{coins}_0 + \sum_{i=1}^{m} \text{coins}_i^{Bob} - \sum_{j=1}^{n} \text{coins}_j^{Alice} \quad .$$

Definition 2 (Base channel).
A base channel *sends exactly one* SUBMIT *message to* $\mathcal{G}_{\text{Ledger}}$ *during the execution of* `openChannel` *and one more during* `closeChannel`*.*

Definition 3 (Virtual channel).
A virtual channel *does not send any* SUBMIT *message to* $\mathcal{G}_{\text{Ledger}}$ *during the execution of* `openChannel`*. Furthermore, if all participating parties in all implicated base channel functionalities are honest, a virtual channel does not send any* SUBMIT *message to* $\mathcal{G}_{\text{Ledger}}$ *during the execution of* `closeChannel` *either, whereas in case one party is dishonest it sends at most one such message.*