
MODULE *Contracts*

This spec captures the behaviour of commitment transactions on the two sides of a Lightning channel.

We model the various kinds of outputs a commitment transactions will have over its lifetime.

The state of the commitment transaction changes in response to the various actions like supersede, publish, etc are taken by parties.

We also do not deal with the communication protocol between nodes for creating and updating commitment transactions. This spec will only focusses on the various commitment transaction and their lifecycle in response to interaction between parties and the blockchain.

We ignore the details of how transactions are signed and just mark transactions as signed. This lets us focus on the specifying the behaviour of the commitment transactions without dealing with lower level complexities.

TODO: We don't track balances yet. Need to do that and don't go into certain states depending on the balance checks

TODO: Add *HTLCs*! Now that will be fun!

EXTENDS *Integers*,
TLC,
Sequences,
FiniteSets

CONSTANTS

<i>CSV</i> ,	The csv value to use in contracts
<i>Height</i> ,	The height up to which we run the spec
<i>NumTxs</i>	The number of commitment txs we want

Current channel contracts only ever have two parties

$Party \triangleq \{ \text{"alice"}, \text{"bob"} \}$

For the first revocation we only need two keys per party

$NumKey \triangleq 2$

Set of all keys

$Key \triangleq \{ \langle p, k \rangle : p \in Party, k \in 0 \dots NumKey \}$

Value to capture missing *CSV* in output

$NoCSV \triangleq \text{CHOOSE } c : c \notin 0 \dots CSV$

Multisig outputs without *CSV* encumbrance

$MultiSig \triangleq Party \times Party \times \{ NoCSV \}$

Multisig outputs with *CSV* encumbrance

$MultiSigWithCSV \triangleq Party \times Party \times \{ CSV \}$

P2WKH outputs, without encumbrance

$$P2WKH \triangleq Key$$

Set of all signatures for all commit txs. The signature in real world is related to the commit transaction. However, we leave out this complication of how the signature is generated. If there is a signature by a key on a tx, it is assumed it is correctly signed as per bitcoin's requirements

$$Sig \triangleq \{\langle p, k \rangle : p \in Party, k \in 0 \dots NumKey - 1\}$$

Value to capture unsigned transactions

$$NoSig \triangleq \text{CHOOSE } s : s \notin Sig$$

$$CT \triangleq [index \mapsto 0 \dots NumTxs, \\ multisig \mapsto MultiSigWithCSV, pk \mapsto P2WKH, \\ local_sig \mapsto Sig \cup \{NoSig\}, \\ remote_sig \mapsto Sig \cup \{NoSig\}]$$

$$PublishId \triangleq \{\langle p, i, h \rangle : p \in Party, i \in 0 \dots NumTxs, h \in 0 \dots Height\}$$

$$NoSpend \triangleq \langle \rangle$$

VARIABLES

<i>alice_cts</i> ,	Commitment tx for <i>alice</i>
<i>bob_cts</i> ,	Commitment tx for bob
<i>alice_brs</i> ,	Breach remedy transactions for <i>alice</i>
<i>bob_brs</i> ,	Breach remedy transactions for bob
<i>mempool_ct</i> ,	The <i>CT</i> that has been broadcasted.
	<i>TODO</i> : Turn into <i>Seq</i> . More than one can be in mempool.
<i>published_ct</i>	The <i>CT</i> that has been included in a block and confirmed.

$$vars \triangleq \langle alice_cts, bob_cts, alice_brs, bob_brs, mempool_ct, published_ct \rangle$$

Helper function to get other party

$$OtherParty(party) \triangleq \text{CHOOSE } p \in Party : p \neq party$$

Create a commitment transaction given the party, index and key to use.

$$CreateCT(party, index, key_num) \triangleq \\ [index \mapsto index, \\ multisig \mapsto \langle party, OtherParty(party), CSV \rangle, \\ pk \mapsto \langle party, key_num \rangle, \\ local_sig \mapsto NoSig, \\ remote_sig \mapsto \langle OtherParty(party), key_num \rangle]$$

$$Init \triangleq \\ \wedge alice_cts = \{CreateCT("alice", 0, 0)\} \\ \wedge bob_cts = \{CreateCT("bob", 0, 0)\} \\ \wedge alice_brs = \{\}$$

$\wedge \text{bob_brs} = \{\}$
 $\wedge \text{mempool_ct} = \text{NoSpend}$
 $\wedge \text{published_ct} = \text{NoSpend}$

$\text{TypeInvariant} \triangleq$
 $\wedge \forall ct \in \text{alice_cts} \cup \text{bob_cts} :$
 $\quad \wedge ct.index \in 0 \dots \text{NumTxs}$
 $\quad \wedge ct.local_sig \in \text{Sig} \cup \{\text{NoSig}\}$
 $\quad \wedge ct.remote_sig \in \text{Sig} \cup \{\text{NoSig}\}$
 $\quad \wedge ct.pk \in P2WKH$
 $\quad \wedge ct.multisig \in \text{MultiSigWithCSV}$
 $\wedge \forall br \in \text{alice_brs} \cup \text{bob_brs} :$
 $\quad \wedge br.index \in 0 \dots \text{NumTxs}$
 $\quad \wedge br.pk \in P2WKH$
 $\wedge \text{mempool_ct} \in \text{PublishId} \cup \{\text{NoSpend}\}$
 $\wedge \text{published_ct} \in \text{PublishId} \cup \{\text{NoSpend}\}$

$\text{MaxIndex}(\text{party_cts}) \triangleq$
 $(\text{CHOOSE } x \in \text{party_cts} : \forall y \in \text{party_cts} : x.index \geq y.index).index$

Create first commitment transactions for given parties

Breach remedy transactions are pre-signed transactions instead of they private key being sent over to the other party.

$\text{SupersedeCommitmentTx}(index) \triangleq$
 $\quad \wedge \text{mempool_ct} \neq \text{NoSpend}$ Stop creating new CTs once FT is spent
 $\quad \wedge$
 $\quad \text{LET } key_index \triangleq 1$
 $\quad \text{IN}$
 $\quad \quad \wedge index > \text{MaxIndex}(\text{alice_cts})$
 $\quad \quad \wedge index > \text{MaxIndex}(\text{bob_cts})$
 $\quad \quad \wedge \text{alice_cts}' = \text{alice_cts} \cup \{\text{CreateCT}(\text{"alice"}, index, key_index)\}$
 $\quad \quad \wedge \text{bob_cts}' = \text{bob_cts} \cup \{\text{CreateCT}(\text{"bob"}, index, key_index)\}$
 $\quad \quad \wedge \text{alice_brs}' = \text{alice_brs} \cup \{[index \mapsto index, pk \mapsto \langle \text{"bob"}, key_index \rangle]\}$
 $\quad \quad \wedge \text{bob_brs}' = \text{bob_brs} \cup \{[index \mapsto index, pk \mapsto \langle \text{"alice"}, key_index \rangle]\}$
 $\quad \wedge \text{UNCHANGED } \langle \text{mempool_ct}, \text{published_ct} \rangle$

Publish a commitment transaction to the blockchain. The commitment is first signed. The protocol allows all commitments to be published, what happens next depends on the status of the commitment transaction.

If the tx is the latest commitment transaction it is succesfully spend.

If not, it gives the other party a chance to spend the breach remedy tx.

TODO: We only spec CSV (self) commitment transaction. We need to handle the non-CSV output being published and co-op closes.

$$\begin{aligned}
& \text{PublishCommitment}(\text{party}, \text{index}, \text{height}) \triangleq \\
& \quad \wedge \text{mempool_ct} = \text{NoSpend} \\
& \quad \wedge \text{mempool_ct}' = \langle \text{party}, \text{index}, \text{height} \rangle \\
& \quad \wedge \text{UNCHANGED} \langle \text{alice_cts}, \text{bob_cts}, \text{alice_brs}, \text{bob_brs}, \text{published_ct} \rangle
\end{aligned}$$

```

*****
Publish a breach remedy transaction in response to a commitment transaction.
party is publishing the breach remedy tx when it is on index CT, and the chain is on height.
This tx is immediately published on chain.

TODO: We skip the BR going through the mempool and confirm it immediately. This can be
improved too.
*****

```

$$\begin{aligned}
& \text{PublishBR}(\text{party}, \text{index}, \text{height}) \triangleq \\
& \quad \text{LET } \text{cts} \triangleq \text{IF } \text{party} = \text{"alice"} \text{ THEN } \text{alice_cts} \text{ ELSE } \text{bob_cts} \\
& \quad \text{IN} \\
& \quad \quad \wedge \text{published_ct} = \text{NoSpend} & \text{No CT is confirmed on chain yet} \\
& \quad \quad \wedge \text{mempool_ct} \neq \text{NoSpend} & \text{Only if some CT has been published} \\
& \quad \quad \wedge \text{mempool_ct}[1] = \text{OtherParty}(\text{party}) & \text{CT was published by the other party} \\
& \quad \quad \wedge \text{mempool_ct}[2] < \text{MaxIndex}(\text{cts}) & \text{Revoked CT was published} \\
& \quad \quad \wedge \text{mempool_ct}[2] = \text{index} & \text{We need to use the BR from the same index} \\
& \quad \quad \wedge \text{height} - \text{mempool_ct}[2] < \text{CSV} & \text{Can only publish BR if CSV hasn't expired} \\
& \quad \quad \wedge \text{published_ct}' = \langle \text{party}, \text{index}, \text{height} \rangle & \text{Record which index was published at what height} \\
& \quad \quad \wedge \text{UNCHANGED} \langle \text{alice_cts}, \text{bob_cts}, \text{alice_brs}, \text{bob_brs}, \text{mempool_ct} \rangle
\end{aligned}$$

$$\begin{aligned}
& \text{Next} \triangleq \\
& \quad \vee \exists i \in 0 \dots \text{NumTxs} : \text{SupersedeCommitmentTx}(i) \\
& \quad \vee \exists i \in 0 \dots \text{NumTxs}, p \in \text{Party}, h \in 0 \dots \text{Height} : \text{PublishCommitment}(p, i, h) \\
& \quad \vee \exists i \in 0 \dots \text{NumTxs}, p \in \text{Party}, h \in 0 \dots \text{Height} : \text{PublishBR}(p, i, h)
\end{aligned}$$

$$\text{Spec} \triangleq \text{Init} \wedge \Box[\text{Next}]_{\langle \text{vars} \rangle}$$