
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 focus 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.

The model defines the initial balance from *alice* to bob. TLA+ will handle situations where channels are balanced and when all the balance is on the other side.

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
<i>InitialBalance</i>	Initial balances for <i>alice</i> and bob

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

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

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

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

$NoSpend \triangleq \langle \rangle$

VARIABLES

$alice_cts,$	Commitment tx for <i>alice</i>
$bob_cts,$	Commitment tx for bob
$alice_brs,$	Breach remedy transactions for <i>alice</i>
$bob_brs,$	Breach remedy transactions for bob
$mempool_ct,$	The <i>CT</i> txs that have been broadcasted.
$published_ct$	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.

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

$Init \triangleq$

Once sided channel to start with

$$\begin{aligned} \wedge \text{alice_cts} &= \{ \text{CreateCT}(\text{"alice"}, 0, 0, \text{InitialBalance}) \} \\ \wedge \text{bob_cts} &= \{ \text{CreateCT}(\text{"bob"}, 0, 0, 0) \} \\ \wedge \text{alice_brs} &= \{ \} \\ \wedge \text{bob_brs} &= \{ \} \\ \wedge \text{mempool_ct} &= \{ \} \\ \wedge \text{published_ct} &= \text{NoSpend} \end{aligned}$$

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

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

$$\begin{aligned} \text{LastCT}(\text{party_cts}) &\triangleq \\ &\text{CHOOSE } ct \in \text{party_cts} : \forall y \in \text{party_cts} : ct.index \geq y.index \end{aligned}$$

Create commitment transaction as well as the corresponding beach remedy txs.

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

delta is the balance going from *alice* to bob. We allow negative balances to enable payments in other other direction.

Parties are free to keep creating *CT* even if *FT* is spent. They will not be usable, but the protocol does not disallow this.

$$\begin{aligned} &\text{SupersedeCommitmentTx}(index, delta) \triangleq \\ &\quad \wedge \\ &\quad \text{LET} \\ &\quad \quad key_index \triangleq 1 \\ &\quad \quad last_alice_ct \triangleq \text{LastCT}(\text{alice_cts}) \\ &\quad \quad last_bob_ct \triangleq \text{LastCT}(\text{bob_cts}) \\ &\quad \text{IN} \\ &\quad \quad \wedge index > \text{MaxIndex}(\text{alice_cts}) \\ &\quad \quad \wedge index > \text{MaxIndex}(\text{bob_cts}) \\ &\quad \quad \wedge last_alice_ct.balance - delta > 0 \end{aligned}$$

$$\begin{aligned}
& \wedge \text{last_bob_ct.balance} + \text{delta} > 0 \\
& \wedge \text{alice_cts}' = \text{alice_cts} \cup \\
& \quad \{ \text{CreateCT}(\text{"alice"}, \text{index}, \text{key_index}, \\
& \quad \quad \text{last_alice_ct.balance} - \text{delta}) \} \\
& \wedge \text{bob_cts}' = \text{bob_cts} \cup \\
& \quad \{ \text{CreateCT}(\text{"bob"}, \text{index}, \text{key_index}, \\
& \quad \quad \text{last_alice_ct.balance} + \text{delta}) \} \\
& \wedge \text{alice_brs}' = \text{alice_brs} \cup \\
& \quad \{ [\text{index} \mapsto \text{index}, \text{pk} \mapsto \langle \text{"bob"}, \text{key_index} \rangle] \} \\
& \wedge \text{bob_brs}' = \text{bob_brs} \cup \\
& \quad \{ [\text{index} \mapsto \text{index}, \text{pk} \mapsto \langle \text{"alice"}, \text{key_index} \rangle] \} \\
& \wedge \text{UNCHANGED } \langle \text{mempool_ct}, \text{published_ct} \rangle
\end{aligned}$$

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.

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

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 immeidiately. This can be improved too.

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

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

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