MODULE *SASwap*

EXTENDS *Naturals*, *Sequences*, *TLC*

CONSTANT *BLOCKS_PER_DAY*
ASSUME $BLOCKS\_PER\_DAY > 0$

CONSTANT *MAX_DAYS_TO_CONCLUDE*
ASSUME $MAX\_DAYS\_TO\_CONCLUDE > 4$

CONSTANT *STEALTHY_SEND_POSSIBLE*
ASSUME $STEALTHY\_SEND\_POSSIBLE \in \text{BOOLEAN}$

VARIABLES *blocks*, *block_txs*, *mempool*, *shared_knowledge*, *sigs*

$networkState \triangleq \langle blocks, block\_txs, mempool \rangle$
$fullState \triangleq \langle networkState, shared\_knowledge, sigs \rangle$

Can use this predicate to limit possible state space by
ignoring the states after *MAX_DAYS_TO_CONCLUDE* has passed

$$
\begin{aligned}
&ConcludedInFiniteDays \triangleq \\
&\quad Len(blocks) \leq BLOCKS\_PER\_DAY * MAX\_DAYS\_TO\_CONCLUDE
\end{aligned}
$$

Can use this predicate to limit possible state space by
ignoring the states where nothing was sent/confirmed in a day

$$
\begin{aligned}
&EachDaySomethingIsConfirmed \triangleq \\
&\quad \neg \exists\, bn \in \text{DOMAIN } blocks : \\
&\qquad \forall\, bn\_next \in bn \,..\, bn + BLOCKS\_PER\_DAY - 1 : \\
&\qquad\quad \land\ bn\_next \in \text{DOMAIN } blocks \\
&\qquad\quad \land\ blocks[bn\_next] = \{\}
\end{aligned}
$$

Define unique values to avoid using strings everywhere

$Alice \triangleq$ "Alice"
$Bob \triangleq$ "Bob"

$sigAlice \triangleq$ "sigAlice"
$sigBob \triangleq$ "sigBob"
$secretAlice \triangleq$ "secretAlice"
$secretBob \triangleq$ "secretBob"

$tx\_start\_A \triangleq$ "tx_start_A"
$tx\_start\_B \triangleq$ "tx_start_B"
$tx\_success \triangleq$ "tx_success"
$tx\_refund\_1 \triangleq$ "tx_refund_1"
$tx\_revoke \triangleq$ "tx_revoke"
$tx\_refund\_2 \triangleq$ "tx_refund_2"
$tx\_timeout \triangleq$ "tx_timeout"

$tx\_spend\_B \triangleq$ "tx_spend_B"
$tx\_spend\_success \triangleq$ "tx_spend_success"
$tx\_spend\_refund\_1 \triangleq$ "tx_spend_refund_1"
$tx\_spend\_refund\_2 \triangleq$ "tx_spend_refund_2"
$tx\_spend\_timeout \triangleq$ "tx_spend_timeout"

who can produce what signature

$$canSignMap \triangleq [Alice \mapsto \{sigAlice, secretAlice\}, \; Bob \mapsto \{sigBob, secretBob\}]$$

$$all\_sigs \triangleq \text{UNION } \{canSignMap[x] : x \in \text{DOMAIN } canSignMap\}$$

$$ConfirmedTransactions \triangleq \text{UNION } \{blocks[bn] : bn \in \text{DOMAIN } blocks\}$$

$$SentTransactions \triangleq ConfirmedTransactions \cup mempool$$

$$\begin{aligned} dependency\_map \triangleq [&tx\_success \mapsto tx\_start\_A, \\ &tx\_refund\_1 \mapsto tx\_start\_A, \\ &tx\_revoke \mapsto tx\_start\_A, \\ &tx\_refund\_2 \mapsto tx\_revoke, \\ &tx\_timeout \mapsto tx\_revoke, \\ &tx\_spend\_B \mapsto tx\_start\_B, \\ &tx\_spend\_success \mapsto tx\_success, \\ &tx\_spend\_refund\_1 \mapsto tx\_refund\_1, \\ &tx\_spend\_refund\_2 \mapsto tx\_refund\_2, \\ &tx\_spend\_timeout \mapsto tx\_timeout] \end{aligned}$$

$$all\_transactions \triangleq$$

each transacton is either dependant or a dependency

$$\text{DOMAIN } dependency\_map \cup \{dependency\_map[x] : x \in \text{DOMAIN } dependency\_map\}$$

transactions that are mutually exclusive

$$conflicts \triangleq \{\{tx\_success,\ tx\_refund\_1,\ tx\_revoke\},\ \{tx\_refund\_2,\ tx\_timeout\}\}$$

transactions *Alice* initially shares signatures on

$$phase0\_to\_share\_Alice \triangleq \{tx\_revoke,\ tx\_timeout\}$$

transactions *Bob* initially shares signatures on

$$phase0\_to\_share\_Bob \triangleq \{tx\_refund\_1,\ tx\_revoke,\ tx\_refund\_2,\ tx\_timeout\}$$

$$HasDependencies(tx) \triangleq tx \in \text{DOMAIN } dependency\_map$$

$$DependencyConfirmed(tx) \triangleq dependency\_map[tx] \in ConfirmedTransactions$$

$$DependencySent(tx) \triangleq DependencyConfirmed(tx) \lor dependency\_map[tx] \in mempool$$

$$DependencyBlock(tx) \triangleq \text{CHOOSE } bn \in \text{DOMAIN } blocks : dependency\_map[tx] \in blocks[bn]$$

$$NoConflicts(tx) \triangleq \forall\, cfl\_set \in conflicts : \lor\, tx \notin cfl\_set \lor \neg\exists\, cfl\_tx \in cfl\_set \setminus \{tx\} : cfl\_tx \in SentTransactions$$

$$SharedSecrets \triangleq \{x[2] : x \in \{xx \in shared\_knowledge : xx[2] \in \{secretAlice,\ secretBob\}\}\}$$

Signatures currently available to the sender
includes *sigs* made by the sender themself,
and anything from shared knowledge

$$AvailableSigs(tx,\ sender) \triangleq sigs[sender][tx] \cup \{x[2] : x \in \{xx \in shared\_knowledge : xx[1] = tx\}\} \cup SharedSecrets$$

All signatures known for certain transaction.
These go to shared knowledge when the transaction is mined
(unless alredy shared)
This is a simplification, because in general there could be
unpublished signatures when threshold signing is used.

But for this contract, this is OK.

$AllSigsForTx(tx) \triangleq \text{UNION } \{sigs[sender][tx] : sender \in \{Alice,\ Bob\}\}$

Adaptor signatures are modelled as having to supply 3 values
for the signature set, where two values are the *sigs*,
and one value is a secret.
For modelling purposes, the secret acts as just another sig.

$$
\begin{array}{l}
SpendConditionsSatisfied(tx,\ sender) \triangleq \\
\quad \text{LET } HasSigs(ss) \triangleq ss \subseteq AvailableSigs(tx,\ sender) \\
\quad \text{IN} \quad \lor \land tx = tx\_start\_A \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice\}) \\
\qquad\qquad \lor \land tx = tx\_start\_B \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigBob\}) \\
\qquad\qquad \lor \land tx = tx\_success \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob,\ secretBob\}) \\
\qquad\qquad \lor \land tx = tx\_refund\_1 \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob,\ secretAlice\}) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq BLOCKS\_PER\_DAY \\
\qquad\qquad \lor \land tx = tx\_revoke \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob\}) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq BLOCKS\_PER\_DAY * 2 \\
\qquad\qquad \lor \land tx = tx\_refund\_2 \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob,\ secretAlice\}) \\
\qquad\qquad\quad\ \ \land DependencyConfirmed(tx) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq DependencyBlock(tx) + BLOCKS\_PER\_DAY \\
\qquad\qquad \lor \land tx = tx\_timeout \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice,\ sigBob\}) \\
\qquad\qquad\quad\ \ \land DependencyConfirmed(tx) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq DependencyBlock(tx) + BLOCKS\_PER\_DAY * 2 \\
\qquad\qquad \lor \land tx = tx\_spend\_B \\
\qquad\qquad\quad\ \ \land HasSigs(\{secretAlice,\ secretBob\}) \\
\qquad\qquad \lor \land tx = tx\_spend\_success \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigBob\}) \\
\qquad\qquad \lor \land tx = tx\_spend\_refund\_1 \\
\qquad\qquad\quad\ \ \land HasSigs(\{sigAlice\}) \\
\qquad\qquad\quad\ \ \land DependencyConfirmed(tx) \\
\qquad\qquad\quad\ \ \land Len(blocks) \geq DependencyBlock(tx) + BLOCKS\_PER\_DAY
\end{array}
$$

$$
\begin{array}{l}
\lor\ \land\ tx = tx\_spend\_refund\_2 \\
\quad \land\ HasSigs(\{sigAlice\}) \\
\lor\ \land\ tx = tx\_spend\_timeout \\
\quad \land\ HasSigs(\{sigBob\})
\end{array}
$$

Unclear what the result of this tx should be

$\lor\ \land\ tx = tx\_spend\_refund\_1\_cooperative$

$$
\begin{array}{l}
CanEnterMempool(tx,\ sender)\ \triangleq \\
\quad \land\ tx \notin SentTransactions \\
\quad \land\ NoConflicts(tx) \\
\quad \land\ \lor\ HasDependencies(tx) \land DependencySent(tx) \\
\qquad \lor\ \neg HasDependencies(tx) \\
\quad \land\ SpendConditionsSatisfied(tx,\ sender)
\end{array}
$$

$$
Share(knowledge)\ \triangleq\ shared\_knowledge' = shared\_knowledge \cup knowledge
$$

Note that the *sigs* record-of-records may not be the most elegant
data structure for the purpose, might be worth it to explore other options

$$
\begin{array}{l}
SignTx(tx,\ ss,\ signer)\ \triangleq \\
\quad \land\ \forall\, s \in ss : s \in canSignMap[signer] \\
\quad \land\ sigs' = [sigs\ \text{EXCEPT} \\
\qquad\qquad ![signer] = [sigs[signer]\ \text{EXCEPT} \\
\qquad\qquad\qquad\qquad ![tx] = sigs[signer][tx] \cup ss]]
\end{array}
$$

$$
\begin{array}{l}
SendTx(tx,\ sender)\ \triangleq \\
\quad \land\ CanEnterMempool(tx,\ sender) \\
\quad \land\ mempool' = mempool \cup \{tx\} \\
\quad \land\ Share(\{\langle tx,\ s\rangle : s \in sigs[sender][tx]\})
\end{array}
$$

Give tx directly to miner, bypassing global *mempool*

$$
\begin{array}{l}
StealthySendTx(tx,\ sender)\ \triangleq \\
\quad \land\ STEALTHY\_SEND\_POSSIBLE \\
\quad \land\ CanEnterMempool(tx,\ sender) \\
\quad \land\ block\_txs' = block\_txs \cup \{tx\}
\end{array}
$$

If participant has a transaction ready to be sent,
they can send it to *mempool*, or stealthy to the miner
(if $STEALTHY\_SEND\_POSSIBLE$ is TRUE)

$$
SendSomething\ \triangleq
$$

$\exists\, sender \in \{Alice,\ Bob\} :$
$\quad \exists\, tx \quad \in all\_transactions :$
$\quad\quad \lor \land SendTx(tx,\ sender)$
$\quad\quad\quad \land$ UNCHANGED $block\_txs$
$\quad\quad \lor \land StealthySendTx(tx,\ sender)$
$\quad\quad\quad \land$ UNCHANGED $\langle mempool,\ shared\_knowledge \rangle$

Conditions to divide the action into phases according to original spec

$Phase\_3\_cond \triangleq tx\_start\_B \in ConfirmedTransactions$
$Phase\_2\_cond \triangleq tx\_start\_A \in ConfirmedTransactions$
$Phase\_1\_cond \triangleq$
$\quad \land \forall\, tx \in phase0\_to\_share\_Alice :$
$\quad\quad \exists\, signed\_pair \in shared\_knowledge : signed\_pair = \langle tx,\ sigAlice \rangle$
$\quad \land \forall\, tx \in phase0\_to\_share\_Bob :$
$\quad\quad \exists\, signed\_pair \in shared\_knowledge : signed\_pair = \langle tx,\ sigBob \rangle$

$InPhase\_3 \triangleq$
$\quad \land Phase\_3\_cond$

$InPhase\_2 \triangleq$
$\quad \land Phase\_2\_cond$
$\quad \land \neg Phase\_3\_cond$

$InPhase\_1 \triangleq$
$\quad \land Phase\_1\_cond$
$\quad \land \neg Phase\_2\_cond$
$\quad \land \neg Phase\_3\_cond$

$InPhase\_0 \triangleq$
$\quad \land \neg Phase\_1\_cond$
$\quad \land \neg Phase\_2\_cond$
$\quad \land \neg Phase\_3\_cond$

Participant actions

helper operators to declutter the action expressions

$NoSigning \triangleq$ UNCHANGED $sigs$

$NothingShared \triangleq$ UNCHANGED $shared\_knowledge$

$AliceAction \triangleq$

LET $Sign(tx, ss) \triangleq SignTx(tx, ss, Alice)$

IN $\lor \land InPhase\_0$
$\land Share(\{\langle tx, sigAlice\rangle : tx \in phase0\_to\_share\_Alice\})$
$\land NoSigning$
$\lor \land InPhase\_1$
$\land Sign(tx\_start\_A, \{sigAlice\})$
$\land NothingShared$
$\lor \land InPhase\_2$
$\land$ Do nothing or refund, if timelock allows
$\lor NoSigning$
$\lor Sign(tx\_refund\_1, \{sigAlice, secretAlice\})$
$\land NothingShared$
$\lor \land InPhase\_3$
$\land \lor \land Share(\{\langle tx\_success, sigAlice\rangle\})$
$\land NoSigning$
$\lor$ Can revoke on short path until $tx\_success$ is shared
$\land \langle tx\_success, sigAlice\rangle \notin shared\_knowledge$
$\land Sign(tx\_refund\_1, \{sigAlice, secretAlice\})$
$\land NothingShared$
$\lor \land \lor Sign(tx\_spend\_B, \{secretAlice\})$
$\lor Sign(tx\_refund\_2, \{sigAlice, secretAlice\})$
$\lor Sign(tx\_spend\_refund\_1, \{sigAlice, secretAlice\})$
$\lor Sign(tx\_spend\_refund\_2, \{sigAlice, secretAlice\})$
$\land NothingShared$

$BobAction \triangleq$

LET $Sign(tx, ss) \triangleq SignTx(tx, ss, Bob)$

IN $\lor \land InPhase\_0$
$\land Share(\{\langle tx, sigBob\rangle : tx \in phase0\_to\_share\_Bob\})$

$\land$ *NoSigning*
$\lor$ $\land$ *InPhase_1* No specific actions
$\land$ *NoSigning*
$\land$ *NothingShared*
$\lor$ $\land$ *InPhase_2*
$\land$ *Sign*(*tx_start_B*, {*sigBob*})
$\land$ *NothingShared*
$\lor$ $\land$ *InPhase_3*
$\land$ $\lor$ *Sign*(*tx_spend_B*, {*secretBob*})
$\lor$ *Sign*(*tx_success*, {*sigBob*, *secretBob*})
$\lor$ *Sign*(*tx_spend_success*, {*sigBob*})
$\lor$ *Sign*(*tx_timeout*, {*sigBob*})
$\lor$ *Sign*(*tx_spend_timeout*, {*sigBob*})
$\land$ *NothingShared*

IncludeTxIntoBlock $\triangleq$
$\land \exists\, tx \in mempool :$
$\land$ IF *HasDependencies*(*tx*)
THEN $\lor$ *DependencyConfirmed*(*tx*)
$\lor$ *dependency_map*[*tx*] $\in$ *block_txs*
ELSE TRUE
$\land$ *block_txs*$'$ = *block_txs* $\cup$ {*tx*}
$\land$ UNCHANGED $\langle$*blocks*, *mempool*, *shared_knowledge*$\rangle$

MineTheBlock $\triangleq$
$\land$ *blocks*$'$ = *Append*(*blocks*, *block_txs*)
$\land$ *mempool*$'$ = *mempool* $\setminus$ *block_txs*
$\land$ *Share*(UNION {{$\langle$*tx*, *s*$\rangle$: *s* $\in$ *AllSigsForTx*(*tx*)} :
tx $\in$ *block_txs* $\setminus$ *mempool*})
$\land$ *block_txs*$'$ = {}

MinerAction $\triangleq$ *IncludeTxIntoBlock* $\lor$ *MineTheBlock*

Invariants

$$
\begin{array}{l}
\mathit{TypeOK} \triangleq \\
\quad \land \quad \forall\, \mathit{pair} \in \mathit{shared\_knowledge} : \\
\qquad\quad \land\, \mathit{pair}[1] \in \mathit{all\_transactions} \\
\qquad\quad \land\, \mathit{pair}[2] \in \mathit{all\_sigs} \\
\quad \land \quad \text{DOMAIN } \mathit{sigs} = \{\mathit{Alice},\ \mathit{Bob}\} \\
\quad \land \quad \forall\, \mathit{sender} \in \text{DOMAIN } \mathit{sigs} : \\
\qquad\quad \forall\, \mathit{tx} \in \text{DOMAIN } \mathit{sigs}[\mathit{sender}] : \\
\qquad\quad\quad \land\, \mathit{tx} \in \mathit{all\_transactions} \\
\qquad\quad\quad \land\, \mathit{sigs}[\mathit{sender}][\mathit{tx}] \subseteq \mathit{all\_sigs}
\end{array}
$$

$$
\begin{array}{l}
\mathit{ConsistentPhase} \triangleq \\
\quad \text{LET } \mathit{phases} \triangleq \langle \mathit{InPhase\_0},\ \mathit{InPhase\_1},\ \mathit{InPhase\_2},\ \mathit{InPhase\_3} \rangle \\
\quad \text{IN} \quad \land\, \exists\, i \in \text{DOMAIN } \mathit{phases} : \mathit{phases}[i] \\
\qquad\quad \land\, \forall\, i \in \text{DOMAIN } \mathit{phases} : \\
\qquad\qquad \text{IF } \mathit{phases}[i] \\
\qquad\qquad\ \text{THEN } \neg \exists\, \mathit{ii} \in \text{DOMAIN } \mathit{phases} : \mathit{ii} \neq i \land \mathit{phases}[\mathit{ii}] \\
\qquad\qquad\ \text{ELSE TRUE}
\end{array}
$$

$$
\begin{array}{l}
\mathit{NoConcurrentSecretKnowledge} \triangleq \\
\quad \lor\, \mathit{tx\_spend\_B} \in \mathit{SentTransactions} \\
\quad \lor\, \neg(\{\mathit{secretAlice},\ \mathit{secretBob}\} \subseteq \mathit{SharedSecrets})
\end{array}
$$

Can use this invariant to check if certain state can be reached
If the *CounterExample* invariant is violated, then the state
has been reached.

$$\mathit{CounterExample} \triangleq \text{TRUE} \land \ldots$$

Temporal properties, Not tested at the moment

$$
\begin{array}{l}
\mathit{AllEventuallyConfirmed} \triangleq \\
\quad \forall\, \mathit{tx} \in \mathit{all\_transactions} : \Diamond(\mathit{tx} \in \mathit{ConfirmedTransactions})
\end{array}
$$

$$
\begin{array}{l}
\mathit{SuccessTxLeadsToSpentB} \triangleq \\
\quad \mathit{tx\_success} \in \mathit{SentTransactions} \leadsto \mathit{tx\_spend\_B} \in \mathit{ConfirmedTransactions}
\end{array}
$$

High-level spec

$$
\begin{array}{l}
\mathit{Init} \triangleq \\
\quad \land\ \mathit{blocks} = \langle\rangle \\
\quad \land\ \mathit{block\_txs} = \{\} \\
\quad \land\ \mathit{mempool} = \{\} \\
\quad \land\ \mathit{shared\_knowledge} = \{\} \\
\quad \land\ \mathit{sigs} = [\mathit{Alice} \mapsto [\mathit{tx} \in \mathit{all\_transactions} \mapsto \{\}], \\
\qquad\qquad\ \ \mathit{Bob} \mapsto [\mathit{tx} \in \mathit{all\_transactions} \mapsto \{\}]]
\end{array}
$$

$$
\begin{array}{l}
\mathit{Next} \triangleq \\
\quad \land\ \mathit{ConcludedInFiniteDays} \\
\quad \land\ \mathit{EachDaySomethingIsConfirmed} \\
\quad \land\ \lor\ \mathit{AliceAction} \quad \land \text{UNCHANGED } \mathit{networkState} \\
\qquad \lor\ \mathit{BobAction} \quad \land \text{UNCHANGED } \mathit{networkState} \\
\qquad \lor\ \mathit{SendSomething} \land \text{UNCHANGED } \langle \mathit{blocks}, \mathit{sigs} \rangle \\
\qquad \lor\ \mathit{MinerAction} \quad \land \text{UNCHANGED } \mathit{sigs}
\end{array}
$$

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{fullState}} \quad \land \text{WF\_}\mathit{networkState}(\mathit{Next})$$