Algorithm 1 Main loop for a node in the proposed fault-tolerant mix-net architecture.

Input: $pki, maxRound, roundTime, peer, baseRole$

1: $state \leftarrow \varnothing$ ➤ Initialize this node's control structure to empty.
2: **loop** ➤ Run until process is stopped by user.
3: $isClient, isEntry, isExit \leftarrow$ PrepareEpoch($state, baseRole, peer$) ➤ See Alg. 2.
4: **if** $isClient$ **then**
5: $successful \leftarrow \top$ ➤ Initialize indicators that decide which payload to send.
6: $isSecondTx \leftarrow \bot$
7: **while** $(r \leftarrow 1 \ldots maxRound) \wedge$ epoch not aborted **do**
8: $successful, isSecondTx \leftarrow$ClientSendAndReceive($state, e, r, successful, isSecondTx$) ➤ See Alg. 3.
9: **end while**
10: **else**
11: $state.FirstPool \leftarrow$ genCoverMsgs($state.Clients^{(e)}$) ➤ Onion-encrypted (cf. lines 6–18, Alg. 3).
12: $state.NextPool \leftarrow$ genCoverMsgs($state.Clients^{(e)}$) ➤ Onion-encrypted (cf. lines 6–18, Alg. 3).
13: $state.SecPool, state.ThirdPool, state.OutPool \leftarrow \varnothing$
14: **while** $(r \leftarrow 1 \ldots maxRound) \wedge$ epoch not aborted **do**
15: $roundTimer \leftarrow roundTime$
16: MixProcessRound($state, e, r, roundTimer, isEntry, isExit$) ➤ See Alg. 4.
17: **end while**
18: **end if**
19: **end loop**