Olympus Module

Note : X_y indicates that y has digitally signed the X using the private keys.
Note : Decryption of keys implies that the digital signatures are being verified using the public keys.
Note: Hash(X) means that it is the cryptographic has function of the variable X.
Note: "Keys" include both public keys and private keys. Public keys are broadcasted to everyone whereas private keys are given to respective elements.

1. Olympus: On receiving(request,"Configuration") from Client:
   //Creates the new configuration and generates the keys and forwards them to the //client and replicas
   Configuration,N = Generate_config(Null)
   Send(response,Configuration) to Client
   Send(response,Configuration) to Replicas
   Send(response,N) to head
   Keys = Generate_keys( )
   Send(response,Keys) to Client
   Send(response,Keys) to Replicas

2. Olympus : On receiving(request,re-configuration,Configuration):
   //Send the wedge request to all the replicas and receives them
   Send(request,$Wedge\_request_{Olympus}$) to Replicas
   Receive(response,Wedge_statements)
   Q_c, Ch = Select_quorums(Wedge_statement)
   For Replica in Qc :
      Send(request,"Running_state") to Replica
      Receive(response,Running_state) from Replica
      //Compares the cryptographic hash of the receive running state with the //cryptographic hash of the running state of longest history replica (Ch)
      If Hash(Running_state) == Ch:
         //The running state is appended into the Inithist Message and the //Olympus sends it after digitally signing it
         Configuration = Generate_config($Inithist_{Olympus}$)
         Send(response,Configuration) to Client
         Send(response,Configuration) to Replicas
         Keys = Generate_keys( )
         Send(response,Keys) to Client
         Send(response,Keys) to Replicas

```
Def  Generate_config(inithist_Olympus):
        C = Assign 2t+1 Replicas
        // N is the number of slots after which the checkpointing has to be performed
        N = Select a random Positive Integer
        For each Replica in C:
                Replica.Mode = Active
                If Running_state:
                        //for Reconfiguration
                        Replica.history = inithist.Running_state.history
                Else:
                        //for initial configuration
                        Replica.history = { }
        Return C,N
```

```
Def  Select_Quorums(Wedge_statements) :
        //Select a set of (t + 1) replicas from Configuration
        //Q_c denotes the set of Replicas called Quorum
        Q_c = Select any (t+1) Replicas from Configuration
        //Checks the consistency of the wedge statements from replicas and
        //compares the histories of different replicas
        If Check_consistency(Q_c, Wedge_statements):
                //Compute the longest history from a Quorum
                Lh = longest_history(Q_c)
                W = Wedge_statements
                //Each replica performs the remaining operations from the catch_up
                and //sends its response as the caught_up
                For each Replica in Q_c :
                        //Catch_up is the list of operations that are present in the
                        longest //history but not in the Replica history
                        Catch_up_Replica = Lh – W_Replica.history
                        Send(request,Catch_up_Replica) to Replica
                        Receive(response,Caught_up_Replica) from Replica
                For Replica1, Replica 2 in Q_c :
                        If Caught_up_Replica1 != Caught_up_Replica2 :
                                Select_Quorums(Wedge_statements)
```

```
        Return Qc,Caught_upReplica_lh
    Else :
        Select_Quorums(Wedge_statements)

Def Check_consistance(Qc , Wedge_statements):

    //Checks all the pairs of replicas such that for a given slot there should be a
    //unique operation
    For Replica1, Replica2 in Qc:
        If <s,o> in Replica1.history and <s,o'> in Replica2.history:
            If o != o' :
                Select_Quorums(Wedge_statements)

    Return True
```