Replica 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. Replicas : On receiving(response,”configuration”) and (response,”keys”) from Olympus

```
//replicas save the configuration and keys
Save (“configuration”,”keys”)
```

2. Head : On receiving (request, o ,i_c,”initial_transmission”) from client or head iteslf

```
Decrypt the request using the keys
If Decrypt fails:
        Ignore the request from the client
Else:
        //Assigns a slot to the operation o
        Assign (s, to=o)
        //N is the number of slots after which checkpoint has to be implemented and is sent by olympus
        //Head forwards the checkpoint proof to the next replicas in the chain
        For every N:
                Checkpoint_proof = []
                Checkpoint = <checkpoint,hash(running_state)>
                Checkpoint_proof = checkpoint_proof U checkpoint
                Send(request,checkpoint_proof)$_{head}$ to the next_replica
        //Creates a shuttle
        Order_proof = [ ]
        Result_prrof = [ ]
        Shuttle = (order_proof,result_proof)
        //Validates the order proof in the shuttle and if it fails sends a reconfiguration request as a //proof of misbehaviour
        If(Validate_proof((s,o),history,Configuration)) :
                //Evaluates the result
                r = Evaluate(o)
                //Append order_proof and result_proof to the shuttle
                order_proof = order_prrof U <order,s,o>$_{head}$
                history = history U [order_proof]
                result_proof = result_prrof U <result,o,Hash(r),i_c>$_{head}$
                Shuttle = (order_proof,result_proof)
                //Send the shuttle to the next replica
                Send(request, o ,i_c , s, shuttle)$_{head}$ to next_replica
```

3. Head : On receiving(response,checkpoint_proof)

```
Decrypt the response using the keys
If Decrypt fails:
        Send(request,"Re-configuration",Configuration) to Olympus
Removes the history prefix to the corresponding checkpoint
```

4. Intermediate replicas : On receiving(request, o ,i_c , s, shuttle) :

```
Decrypt the request from the previous_replica using the keys
If Decrypt fails:
        Send(request,"Re-configuration",Configuration) to Olympus
//Validates the order proof in the shuttle
If (Validate_proof((s,o),history,Configuration) :
        //Evaluates the result
        r = Evaluate(o)
        //Append order_proof and result_proof to the shuttle
        order_proof = order_prrof U <order,s,o>$_{replica}$
        history = history U [order_proof]
        result_proof = result_prrof U <result,o,Hash(r),i_c>$_{replica}$
        Shuttle = (order_proof,result_proof)
        //Send the shuttle to the next replica
        Send(request, o ,i_c , s, shuttle)$_{replica}$ to next_replica
```

5. Intermediate replicas : On receiving (request,checkpoint_proof)

```
Decrypt the request using the keys
If Decrypt fails:
        Send(request,"Re-configuration",Configuration) to Olympus
//All the replicas append the checkpoint and their current running state to the
//checkpoint_point proof and send it to the next replicas
Checkpoint = <checkpoint,hash(running_state)>
Checkpoint_proof = checkpoint_proof U checkpoint
Send(request,checkpoint_proof)$_{Replica}$ to the next_replica
```

6. Intermediate Replicas : On receiving(response,checkpoint_proof)

```
Decrypt the request from the client using the keys
If Decrypt fails:
        Send(request,"Re-configuration",Configuration) to Olympus
Removes the history prefix to the corresponding checkpoint
Send(response,checkpoint_proof)$_{Replica}$ to the previous_replica
```

7. Tail : On receiving(request,o,i_c,s,shuttle)

```
Decrypt the request from the previous_replicas using the keys
If Decrypt fails:
        Send(request,"Re-configuration" ,Configuration) to Olympus
//Validates the order proof in the shuttle
If (Validate_proof((s,o),history,Configuration)) :
        //Evaluates the result
        r = Evaluate(o)
        //Append order_proof and result_proof to the shuttle
        order_proof = order_prrof U <order,s,o>$_{tail}$
        history = history U [order_proof]
        result_proof = result_proof U <result,o,Hash(r),i_c>$_{tail}$
        //Tail sends the result proof and result to the client and the
        Send(response,result_proof,r)$_{tail}$ to client
        Result_shuttle = result_proof
        Send(response,Result_shuttle)$_{Tail}$ to previous_replicas

8.  Tail : On receiving (request,checkpoint_proof)
    Decrypt the request using the keys
    If Decrypt fails:
        Send(request,"Re-configuration" ,Configuration) to Olympus
    Checkpoint = <checkpoint,hash(running_state)>
    Checkpoint_proof = checkpoint_proof U checkpoint
    Removes the history prefix to the corresponding checkpoint
    Send(response,checkpoint_proof)$_{Tail}$ to the previous_replica

9.  Replicas : On receiving(response, result_shuttle)
    Decrypt the response using the keys
    If Decrypt fails:
        Send(request,"Re-configuration",Configuration) to Olympus
    //cache the result shuttle and sends it back to the previous replicas along the chain
    Save(result_shuttle,r)
    Send(response,result_shuttle)$_{Replica}$ to previous replicas

10. Replicas : On receiving(request,o,i_c,"retransmission") from client
    Decrypt the request from the client using the keys
    If Decrypt fails:
        Ignore the request from the client
    Else:
        //If the correct replica has the shuttle then it sends the shuttle back to the client
        If (o,i_c) in Result_shuttle:
            Send(response,Result_shuttle,r)$_{replica}$ to client
        // If the replica is in immutable state, then it sends an error message to the client
        Elif replica.mode == Immutable :
            Send(response,error)$_{replica}$ to client
        //In all other cases, if replica is not a head, it redirects the request to the head along
        //the chain and starts a timer
        Else:
            If replica is not head:
                Send(o,i_c)$_{replica}$ to previous_replica
                Timer.start()
                //waits for the shuttle to arrive or the timer to expire
                Await(result_shuttle or timer.expires())
                If timer.expires():
                    Send(request,re-configuration,Configuration) to Olympus
                Elif replica has result_shuttle:
```

```
            Send(response,result_shuttle,r)$_{\text{replica}}$ to client
            Timer.stop()

Else:
// If the head has the result shuttle cached corresponding to the i_c, it sends
//the result shuttle to the client
    If (o,i_c) in Result_shuttle:
            Send(response,Result_shuttle,r)$_{\text{head}}$ to client
//If head has the <s,o> pair in it's order proof, it starts a timer and waits for
//the result shuttle
    Elif o in <order_proof>$_{\text{head}}$ :
            Timer.start()
            //waits for the shuttle to arrive or the timer to expire
            Await(shuttle or timer.expires())
            If timer.expires():
                    Send(request,re-configuration,Configuration) to Olympus
            Elif head has result_shuttle:
                    Send(response,result_shuttle,r)$_{\text{head}}$ to client
                    Timer.stop()
    Else:
            //The head doesn't have an order proof corresponding to
            //operation o
            Send(request,o,i_c) to head
            Timer.start()
            //waits for the Result_shuttle to arrive or the timer to expire
            Await(Result_shuttle or timer.expires())
            If timer.expires():
                    Send(request,re-configuration,Configuration) to Olympus
            Elif head has result_shuttle:
                    Send(response,result_shuttle,r)$_{\text{head}}$ to client
                    Timer.stop()
```

11. Replicas: On receiving(request,Wedge_request$_{\text{Olympus}}$) from Olympus

```
Decrypt the request using the keys
If Decrypt fails:
        Send(request,"Re-configuration" ,Configuration) to Olympus
//If they are active, they become immutable and send their current history along with
//checkpoint_proof in the form of a wedge_statement to the Olympus
If Replica.mode = Active:
        Replica.mode = Immutable
Wedge_statement = Checkpoint_proof U history
Send(response,Wedge_statement)$_{\text{Replica}}$ to Olympus
```

12. Replicas: On receiving(request,"Catch_up") from Olympus

```
Decrypt the request using the keys
If Decrypt fails:
        Send(request,”Re-configuration” ,Configuration) to Olympus
//Executes all the remaining operations missing from the longest history
State = Execute(Chatch_up)
Caught_up = Hash(State)
Send(response,Caught_up)$_{Replica}$ to Olympus
```

13. Replicas : On receiving(request,”Running_state”) from Olympus

```
//Get the current running state of the replica and send it to the Olympus
Running_state = get_running_state( )
Send(response,Running_state)$_{Replica}$ to Olympus
```

```
Def Validate_proof((s,o),history,C):
        If Replica in C:
                If Replica.mode = Active :
                        For (s,o`) in Replica.history:
                                If o!=o` :
                                        Send(request,”Re-Configuration” ,Configuration)
                        Return True
```