Appendix

```
/* Public signals                                */
signal input: node_ihr
signal input: ihr_hash
/* Private signals                               */
signal input: salt
signal input: required_ihr
/* Output signal                                 */
signal output: if_pass
/* Range proof check                             */
signal buffer
signal range_check
if node_ihr > required_ihr - buffer && node_ihr < required_ihr
 + buffer then
 | range_check = true
end
/* Verify hash                                   */
signal hash
signal hash_check
/* RIPEMD160 to calculate the hash               */
hash = RIPEMD160 (salt, required_ihr)
if hash == ihr_hash then
 | hash_check = true
end
if range_check && hash_check then
 | if_pass = true
else
 | if_pass = false
end
```
Algorithm 1: ZK IHR Circuit

```
begin
end
```
Algorithm 2: VDF Proof

Merkle Chain

```
pre: the snip is added to the data
post: the data is added to the chain
begin
   add_node(snip)
   d ← snip
   if head = null then
    | head,tail ← add_data(d)
   else
    | tail ← add_data(d)
   end
end
```
Algorithm 3: class MerkleChain

```
pre: the value is added to the vector
post: the vector is generated to a merkle tree and added to the
       chain
New Vector data
begin
   data ← d
   if size(data) == max_block_size then
    | generate_root(data)
   end
end
```
Algorithm 4: add_data(d)

```
pre: the vector data is added as the leaves
post: merkel tree and its root is generated
New Vector temp_data
begin
   temp_data ← data
   while temp_data > 1 do
   end
   i = 0 i ¡ size(temp_data)i + 2 Left← temp_data[i]; Right ←
    i+1 == size(temp_data)temp_data[i]else
      temp_data[i + 1]end
      combined = Left + Right ; new_temp_data ←
       hash(combined)
      temp_data ← new_temp_data
      node_root ← temp_data[0]
   end
end
```
Algorithm 5: generate_root()

```
initialized: chain is a object of class MerkleChain and string
             data
begin
   while true do
    | Output "enter data (q to quit)"
   end
   Get data
   if d then
    | a
   end
   ta = q else
    | a
   end
   ddnode(data)
end
```
Algorithm 6: main()

```
begin
end
```
Algorithm 7: Ring

```
begin
end
```
Algorithm 8: Gossip

```
begin
end
```
Algorithm 9: Fork Resolve

```
begin
end
```
Algorithm 10: Update Weight

```
begin
end
```
Algorithm 11: Fork Proof

```
begin
end
```
Algorithm 12: Tail Election

```
begin
end
```
Algorithm 13: Block Req

```
begin
end
```
Algorithm 14: Block Req

```
begin
end
```
Algorithm 15: Hash Reward

```
begin
end
```
Algorithm 16: Network Graph

```
begin
end
```
Algorithm 17: Routing Path

```
begin
end
```
Algorithm 18: Stake Requirement

```
begin
end
```
Algorithm 19: Stake Withdrawal

```
begin
end
```
Algorithm 20: Taxes

```
begin
end
```
Algorithm 21: Regulator Script

```
begin
end
```
Algorithm 22: Swap Script

```
begin
end
```
Algorithm 23: Exchange Rate

```
begin
end
```
Algorithm 24: DAO Contracts

```
begin
end
```
Algorithm 25: DAO Payout

```
begin
end
```
Algorithm 26: DAO Dividend

```
begin
end
```
Algorithm 27: DAO removal