REFERENCE ALGORITHMS

Purva Choudhari,* Ajay Joshua†

https://github.com/projectblink

February 22, 2023 (Genesis Version)

Algorithm 1: Network Graph & Path Finding

Algo

Algorithm 2: Onion Peeling

Algo

Algorithm 3: Node Weights

Algo

Algorithm 4: Snip Construction

Algo

Algorithm 5: Hash Proofing

Algo

Algorithm 6: Hash Reward

Algorithm 7: ZK IHR Circuit

```
/* 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 and 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
/* Bandwidth circuit ≡ IHR circuit                         */
```

Algorithm 8: Tax Script

Key: signature, amount, current_exchange_rate, preimage_of_signature, tax_percent

Output: updated stateful contract for the sender new stateful contract for the receiver

begin

DataLen = 1

utxo_amount ← *initial_amount*

pubKey ← pubkey of the sender

exchange_rate ← *initial_exchange_rate*

tds ← *TDS*

Function spend *(sig, amount, current_exchange_rate, tax_percent, receiver_pubkey,preimage)*:

if *checkSig(sig, pubKey)* ***and*** *Tx.checkPreimage(preimage)* **then**

scriptCode ← $SigHash.scriptCode(preimage)$

codeend ← position where the opcode ends

codepart ← $scriptCode$[: $codeend$]

gains ← $(amount * current\_exchange\_rate) - (amount * exchange\_rate)$

if $gains > 0$ **then**

amount ← $amount - (gains * (tax\_percent/100)) * (current\_exchange\_rate)$

if $amount \leq (amount - tds)$ ***and*** $sender == pubKey$ ***and*** $amount \geq 0$ **then**

$utxo\_amount \leftarrow utxo\_amount - amount$ **end**

end

$updated\_script \leftarrow codepart + utxo\_amount + sender + current\_exchange\_rate + tds$

$new\_script \leftarrow codepart + utxo\_amount + receiver\_pubkey + current\_exchange\_rate + tds$

$hash \leftarrow sha256(updated\_script + new\_script)$

if $hash == SigHash.hashOutputs(preimage)$ **then**

$TRUE$

end

end

end

end

Algorithm 10: Open Order Swap Script

Algorithm 11: Exchange Rate Calculation

Algorithm 12: Staking Script

Algo

Algorithm 13: DAO Contracts

Algorithm 14: Token Minting Procedure

Algorithm 15: Transfer and renting fees

```
/* Transfer and renting fees can only be deployed after a stable
   algorithm is written.                                   */
```

*https://github.com/Purva-Chaudhari

†https://github.com/I-Corinthian

Algorithm 9: Merkle Chain

```
class MerkleChain
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
```

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

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

```
main()
initialized: chain is an object of class MerkleChain and string data
begin
    while true do
        Output "enter data (q to quit)" Get data
        if data = q then
            Break
            else
                addnode(data)
            end
        end
    end
end
```