REFERENCE ALGORITHMS

Purva Choudhari,* Ajay Joshua†

https://github.com/projectblink

February 24, 2023 (Genesis Version)

Algorithm 1: Path Finding

```
Function findPath (fromNode, toNode):
    /* Find a route from from Node to to Node                    */
    paths = getAllPaths(fromNode)
    routes = [ ]
    for path in paths do
        /* Check if the path is connected to the destination node
           */
        if path.toNode == toNode then
            return [path]
            /* Try to find a route from the destination node
               through this channel                              */
            route = findPath(path.toNode, toNode)
            if route is not None then
                /* Add this path to the route                    */
                routes.append([path] + route)
            end
        end
        /* Return the route                                      */
        if len(routes) > 0 then
            return (routes)
            else
                return None
            end
        end
    end
end
```

Algorithm 2: Onion Peeling

```
Function onion_path (mint_hash, route):
    /* Get the next hop path in the route                        */
    next_path = route.pop()
    packet = create_onion_packet(mint_hash, next_path)
    for path in reversed(route) do
        eph_key = generate_ephemeral_key()
        packet = add_path_to_onion_route(path, eph_key, packet)
    end
    send_packet_to_next_hop_path(packet, next_path)
    response = receive_response_from_next_hop_path()
    for path in reversed(route) do
        response = decrypt_response_with_ephemeral_key(response,
          path, eph_key)
    end
    return response
end

/* Notes :  The onion peeling algorithm is used to protect the
   privacy of the mint route, by encrypting the mint information
   multiple times, with each layer containing information for the
   next hop.  As the payment packet is passed from hop to hop,
   each node removes a layer of encryption to reveal the next hop
   in the route.

   • mint_hash is the unique identifier for the minted transaction
   • route is a list of the nodes in the mint route
   • add_path_to_onion_packet function adds a new layer to the onion
     packet for the current hop
   • ephemeral key will be used to decrypt the response from that hop

                                                                */
```

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  and hash_check then
    if_pass = true
else
    if_pass = false
end
/* Bandwidth circuit ≡ IHR circuit                              */
```

Algorithm 8: Merkle Chain

```
class MerkleChain
pre: the snip is added to the data
post: the data is added to the chain
add_node(snip)
d ← snip
if head = null then
    head,tail ← add_data(d)
else
    tail ← add_data(d)
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
New Vector data
data ← d
if size(data) == max_block_size then
    generate_root(data)
end

---

generate_root()
pre: the vector data is added as the leaves
post: merkel tree and its root is generated
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]

---

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

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

†https://github.com/I-Corinthian

Algorithm 9: Open Order Swap Script

```
declare token_a as integer
declare seller as PubKey
declare token_b as integer
declare mature_time as integer
set mature_time as expiry_time
Function order (sig, b, buyer, current_exchange_rate_value,
  preimage):
    if mature_time > SigHash.nLocktime(preimage) then
        if checkSig(sig, buyer) then
            if Tx.checkPreimage(preimage) then
                if b == this.token_b then
                    scriptCode = SigHash.scriptCode(preimage)
                    codeend = 104
                    codepart = scriptCode[:104]
                    outputScript_send = codepart + buyer +
                      num2bin(this.token_a, 8) +
                      num2bin(current_exchange_rate_value, 8) +
                      num2bin(tds, 8)
                    output_send =
                      Utils.writeVarint(outputScript_send)
                    outputScript_receive = codepart + this.seller +
                      num2bin(this.token_b, 8) +
                      num2bin(current_exchange_rate_value, 8) +
                      num2bin(tds, 8)
                    output_receive =
                      Utils.writeVarint(outputScript_send)
                    hashoutput =
                      hash256(output_send+output_receive)
                    if hashoutput ==
                      SigHash.hashOutputs(preimage) then
                        /* order is open  placed                */
                    end
                end
            end
        end
    end
end
```

```
Function claim (sig, value, pubKey, current_exchange_rate_value,
  preimage):
    if mature_time < SigHash.nLocktime(preimage) then
        if pubKey == this.seller then
            if checkSig(sig, pubKey) then
                if Tx.checkPreimage(preimage) then
                    if value == this.token_a then
                        scriptCode = SigHash.scriptCode(preimage)
                        codeend = 104
                        codepart = scriptCode[:104]
                        outputScript_claim = codepart + pubKey +
                          num2bin(this.token_a,8) +
                          num2bin(current_exchange_rate_value,8) +
                          num2bin(tds, 8)
                        output_claim =
                          Utils.writeVarint(outputScript_claim)
                        hashoutput = hash256(output_claim)
                        if hashoutput ==
                          SigHash.hashOutputs(preimage) then
                            /* claim is successful              */
                        end
                    end
                end
            end
        end
    end
end
```

Algorithm 10: Exchange Rate Calculation

Algorithm 11: 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
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 ≤ (amount − tds) and sender ==
              pubKey and amount ≥ 0 then
                utxo_amount ← utxo_amount - amount
            end
        end
        updated_script ← codepart + utxo_amount+sender +
          current_exchange_rate + tds
        new_script ← codepart+utxo_amount + receiver_pubkey +
          current_exchange_rate + tds
        hash ← sha256(updated_script+new_script)
        if hash == SigHash.hashOutputs(preimage) then
            true
        end
    end
end
```

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 the stable
   coin algorithm is written.                                    */
```