

8 Prototype Verification for Certificate Issuance (1)

8.1 Account Generation

8.1.1 Account Generation

Using the Cardano Blockchain to Generate Keys. The private and public key pair is used for signing and authenticating transactions.

Prototype : Source Code (gen_account.js)

```
const CardanoWasm = require('@emurgo/cardano-serialization-lib-nodejs');
const crypto = require('crypto');

// Generate 32 bytes of entropy
const entropy = crypto.randomBytes(32);

// Generate a private key (BIP32 format)
const rootKey = CardanoWasm.Bip32PrivateKey.from_bip39_entropy(entropy, '');

// Derive account key
const accountKey = rootKey.derive(1852 | 0x80000000) // Purpose
    .derive(1815 | 0x80000000) // Coin type (ADA)
    .derive(0 | 0x80000000); // Account

// Derive address key and generate address
const privateKey = accountKey.derive(0).derive(0).to_raw_key();
const publicKey = privateKey.to_public();
const baseAddress = CardanoWasm.BaseAddress.new(
    0, // Testnet = 0, Mainnet = 1
    CardanoWasm.StakeCredential.from_keyhash(publicKey.hash()),
    CardanoWasm.StakeCredential.from_keyhash(publicKey.hash())
```

```
).to_address().to_bech32();

// Display private key, public key, and address
console.log("Private Key (Hex):",
Buffer.from(privateKey.as_bytes()).toString('hex'));
console.log("Public Key (Hex):",
Buffer.from(publicKey.as_bytes()).toString('hex'));
console.log("Address (Bech32):", baseAddress);
```

Execution result

```
[ec2-user@ip-172-31-19-190 node-apps]$ node gen_account.js
Private Key (Hex):
08be93a5627776f21d660f1d0aefee6a7368222e15b1b6a45077cdba27cd0948b2f9bd380ddc5fe10be7e94d188338fc697d636b2
Public Key (Hex): 60e5bb3d5c2ed6c1847922e6a03468307239c22d8e88b8a0ce6398a9c60938e5
Address (Bech32):
addr_test1qq39wdxwxek2nrx0uj2ufhya4squcdsqttm3yqjj3xpul8ez2u6vudnv4xxvley4cnwfmtpesmqkhhzgp99zv70s00d88q
```

Private Key (Hex):

08be93a5627776f21d660f1d0aefee6a7368222e15b1b6a45077cdba27cd0948b2f9bd380d
dc5fe10be7e94d188338fc697d636b20a4d9706284f2f3665468a1

Public Key (Hex):

60e5bb3d5c2ed6c1847922e6a03468307239c22d8e88b8a0ce6398a9c60938e5

Address (Bech32):

addr_test1qq39wdxwxek2nrx0uj2ufhya4squcdsqttm3yqjj3xpul8ez2u6vudnv4xxvley4cnwfmtp
esmqkhhzgp99zv70s00d88q

8.2 Transactions and Certificate Issuance

8.2.1 Transaction Creation

Using the Cardano Blockchain, Generate a Transaction with the Generated Private Key and Obtain the Serialized Transaction ID

Prototype : Source Code (gen_simple_tx5.js)

```
const CardanoWasm = require('@emurgo/cardano-serialization-lib-nodejs');
const axios = require('axios');
require('dotenv').config(); // Load environment variables

const privateKeyHex = process.env.PRIVATE_KEY_HEX;
const projectId = process.env.PROJECT_ID;
const apiBase = "https://cardano-preprod.blockfrost.io/api/v0";

//const address =
"addr_test1qq54kguth0dtagn4j5skz3qwcm7tz2qrj8klrllfpfefe2pftv3chw76h638t9fpv9z
qa3huky5q8y0d78l7jznjn5qscstrm";

async function main() {

    const privateKey =
CardanoWasm.PrivateKey.from_extended_bytes(Buffer.from(privateKeyHex, 'hex'));

    // Generate address from private key
    const generatedPublicKey = privateKey.to_public();
    const address = CardanoWasm.BaseAddress.new(
        0, // Testnet = 0, Mainnet = 1
        CardanoWasm.StakeCredential.from_keyhash(generatedPublicKey.hash()),
        CardanoWasm.StakeCredential.from_keyhash(generatedPublicKey.hash())
    );
}
```

```

    ).to_address().to_bech32();

    console.log("address:", address);

    const maxUTXO = await getMaxUTXO(address);
    const txHash = maxUTXO.txHash;
    const txIndex = maxUTXO.txIndex;
    const utxoAmount = parseInt(maxUTXO.amount);
    const fee = 200000; // Transaction fee (Lovelace)
    const sendAmount = utxoAmount - fee; //Transfer amount(after deducting
fees)
    const protocolParams = await getProtocolParameters();
    const txBuilder = CardanoWasm.TransactionBuilder.new(protocolParams);

    txBuilder.add_input(
        CardanoWasm.Address.from_bech32(address),
        CardanoWasm.TransactionInput.new(
            CardanoWasm.TransactionHash.from_bytes(Buffer.from(txHash,
"hex")),
            txIndex
        ),
        CardanoWasm.Value.new(CardanoWasm.BigNum.from_str(utxoAmount.toString(
)))
    );

    txBuilder.add_output(
        CardanoWasm.TransactionOutput.new(
            CardanoWasm.Address.from_bech32(address),
            CardanoWasm.Value.new(CardanoWasm.BigNum.from_str(sendAmount.toStr
ing()))
        )
    );

    txBuilder.set_fee(CardanoWasm.BigNum.from_str(fee.toString()));

```

```

const txBody = txBuilder.build();
const txHex = CardanoWasm.hash_transaction(txBody);
const witnessSet = CardanoWasm.TransactionWitnessSet.new();
const vkeys = CardanoWasm.Vkeywitnesses.new();
vkeys.add(CardanoWasm.Vkeywitness.new(
  CardanoWasm.Vkey.new(privateKey.to_public()),
  privateKey.sign(txHex.to_bytes())
));
witnessSet.set_vkeys(vkeys);

// Build and serialize the transaction
const signedTx = CardanoWasm.Transaction.new(txBody, witnessSet);
const signedTxHex = Buffer.from(signedTx.to_bytes()).toString('hex');
console.log("Signed Transaction (Hex):", signedTxHex);
}

async function getMaxUTXO(address) {
  try {
    const response = await
    axios.get(`${apiBase}/addresses/${address}/utxos`, {
      headers: { 'project_id': projectId }
    });
    const utxos = response.data;
    if (utxos.length === 0) {
      throw new Error("No UTXOs found for this address.");
    }
    // Select the largest UTXO
    let maxUTXO = utxos[0];
    utxos.forEach(utxo => {
      if (parseInt(utxo.amount[0].quantity) >
      parseInt(maxUTXO.amount[0].quantity)) {
        maxUTXO = utxo;
      }
    });
  }
}

```

```

    });
    console.log(maxUTXO);
    return {
      txHash: maxUTXO.tx_hash,
      txIndex: maxUTXO.tx_index,
      amount: maxUTXO.amount[0].quantity
    };
  } catch (error) {
    console.error('Error fetching UTXOs:', error);
    throw error;
  }
}

async function getProtocolParameters() {
  try {
    const response = await
    axios.get(`${apiBase}/epochs/latest/parameters`, {
      headers: {
        'project_id': projectId
      }
    });
  });
  const data = response.data;

  return CardanoWasm.TransactionBuilderConfigBuilder.new()
    .fee_algo(
      CardanoWasm.LinearFee.new(
        CardanoWasm.BigNum.from_str(data.min_fee_a.toString()),
        CardanoWasm.BigNum.from_str(data.min_fee_b.toString())
      )
    )
    .pool_deposit(CardanoWasm.BigNum.from_str(data.pool_deposit))
    .key_deposit(CardanoWasm.BigNum.from_str(data.key_deposit))
    .max_value_size(data.max_val_size)
    .max_tx_size(data.max_tx_size)

```

```

        .coins_per_utxo_word(CardanoWasm.BigNum.from_str(data.coins_per_utxo_word))

        .build();
    } catch (error) {
        console.error('Error fetching protocol parameters:', error);
        throw error;
    }
}

main().catch(console.error);

```

Execution result

```

[ec2-user@ip-172-31-19-190 node-apps]$ node gen_simple_tx5.js
address:
addr_test1qq54kguth0dtagn4j5skz3qwcmtz2qrj8klr1lfpfefe2pftv3chw76h638t9fpv9zqa3huky5q8y0d7817jznjn5qscs
{
  address:
'addr_test1qq54kguth0dtagn4j5skz3qwcmtz2qrj8klr1lfpfefe2pftv3chw76h638t9fpv9zqa3huky5q8y0d7817jznjn5qscs
  tx_hash: 'aa92a3df4bbfacf22108b66f2d6a245002ae8501b51240fb1df8bbab9a759e29',
  tx_index: 0,
  output_index: 0,
  amount: [ { unit: 'lovelace', quantity: '9997800000' } ],
  block: '4a29774fae39f2b8beafcf4812c5432e1f0e78d3a6973fbf111ba8e13f9b1cd6',
  data_hash: null,
  inline_datum: null,
  reference_script_hash: null
}
Signed Transaction (Hex):
84a30081825820aa92a3df4bbfacf22108b66f2d6a245002ae8501b51240fb1df8bbab9a759e2900018182583900295b238bbbdab

```

Signed Transaction (Hex):

84a30081825820aa92a3df4bbfacf22108b66f2d6a245002ae8501b51240fb1df8bbab9a759e2900018182583900295b238bbbdabea275952161440ec6fcb1280391edf1ffe90a729ca8295b238bbbdabea275952161440ec6fcb1280391edf1ffe90a729ca81b0000000253e74500021a00030d40a100818258203745f60982871f6d49d0eb0cf1b02b4eafd128479933311292387df6ed86fcad58404770c9eb8bc2da34356650d35d2ad3e5fa56b1ec6babe31b88204df0f20bbe39941b758d708687898e0d1a683478107f7ddc5afe9df3970efb0f2c6509687f08f5f6

8.2.2 Transaction Sending

Using the Cardano Blockchain to Submit a Transaction.

The transaction is submitted using the transaction ID via the Blockfrost API, and a response is received.

Prototype : Source Code (submit_tx.js)

```
const axios = require('axios');
require('dotenv').config();// Load environment variables

const apiKey = process.env.PROJECT_ID; // Blockfrost API key
const signedTxHex = process.argv[2]; // Get signedTxHex from command-line arguments

if (!signedTxHex) {
  console.error('Error: Please provide the signed transaction hex string as a command line argument. ');
  process.exit(1);
}

// Function to send a transfer transaction
async function sendTransaction() {
  try {
    const response = await axios.post('https://cardano-preprod.blockfrost.io/api/v0/tx/submit', Buffer.from(signedTxHex, 'hex'), {
      headers: {
        'Content-Type': 'application/cbor',
        'project_id': apiKey,
      },
    });

    console.log('Transaction sent successfully:', response.data);
  } catch (error) {
```



```
        console.error('Error sending transaction:', error.response ?
error.response.data : error.message);
    }
}
sendTransaction();
```

Execution result

```
[ec2-user@ip-172-31-19-190 node-apps]$ node submit_tx.js
84a30081825820aa92a3df4bbfacf22108b66f2d6a245002ae8501b51240fb1df8bbab9a759e2900018182583900295b238bbdbab

Transaction sent successfully: c13f37e1bae727e26052fe108fa0a96f5ea02b92ba17f70b969cf24351e18712
```

Transaction sent successfully:

c13f37e1bae727e26052fe108fa0a96f5ea02b92ba17f70b969cf24351e18712

Reference: Check with the Explorer

<https://preprod.cardanoscan.io/transaction/c13f37e1bae727e26052fe108fa0a96f5ea02b92ba17f70b969cf24351e18712>

Tranzaction Detail

トランザクション詳細	
トランザクションID c13f37e1bae727e26052fe108fa0a96f5ea02b92ba17f70b969cf24351e18712	生成時間 Sep 6, 2024 6:10:57 PM
ブロック 2662284	手数料合計 0.1 ADA
確認 High 90071 confirmations	送信合計 9,997.6 ADA
エポック / スロット 165 / 292257	証明書 0
確定スロット 69930657	
内訳 UTXOs	
ウォレット stake_testluq54kguth0dtagn4j5skz3qwc7tz2qj8klrllpfe2qekr6yp	
送信ADA -0.1 ADA	

8.2.3 Transaction Hash

Using the Cardano Blockchain, Retrieve the Transaction ID from the Serialized Transaction.

Prototype : Source Code (hash_tx.js)

```
const serializedTxHex = process.argv[2]; // Get signedTxHex from command-line arguments
const CardanoWasm = require('@emurgo/cardano-serialization-lib-nodejs');

// Convert hexadecimal string to byte array
const txBytes = Buffer.from(serializedTxHex, 'hex');

// Deserialize the transaction
const transaction = CardanoWasm.Transaction.from_bytes(txBytes);

// Get the transaction body
const txBody = transaction.body();

// Get the transaction ID
const txHash = CardanoWasm.hash_transaction(txBody);

// Output the transaction ID in hexadecimal format
console.log("Transaction ID:", Buffer.from(txHash.to_bytes()).toString('hex'));
```

Execution result

```
[ec2-user@ip-172-31-19-190 node-apps]$ node hash_tx.js
84a30081825820aa92a3df4bbfacf22108b66f2d6a245002ae8501b51240fb1df8bbab9a759e2900018182583900295b238bbdbab
Transaction ID: c13f37e1bae727e26052fe108fa0a96f5ea02b92ba17f70b969cf24351e18712
```

Transaction ID:

c13f37e1bae727e26052fe108fa0a96f5ea02b92ba17f70b969cf24351e18712

8.3 Block Information Construction

8.3.1 Research on Block Validation

Using the Cardano Blockchain, Retrieve Block Information from the Latest Block and Use the Block Height as a Key to Retrieve Block Information, then Compare It with the Previous Hash Value.

Prototype : Source Code (block_fetch.mjs)

```
import fetch from 'node-fetch';
import dotenv from "dotenv";

const res = dotenv.config()
const apiKey = process.env.PROJECT_ID; // Blockfrost API key
let lastHeight = 0;
let lastHash = "";

// Function to fetch the latest block information
const fetchBlockInfo = async () => {
  try {
    const response = await fetch('https://cardano-
preprod.blockfrost.io/api/v0/blocks/latest', {
      method: 'GET',
      headers: {
        'project_id': apiKey
      }
    });

    if (response.ok) {
      const blockInfo = await response.json();
      const currentHeight = blockInfo.height;
```

```

        // Execute processing only if the height of the last confirmed
        block is different
        if (lastHeight !== currentHeight) {

            if (lastHeight > 0 && currentHeight > lastHeight + 1) {
                // Fetch information for missing blocks
                for (let h = lastHeight + 1; h < currentHeight; h++) {
                    await fetchBlockDetails(h);
                }
            }
            if (lastHash !== blockInfo.previous_block) {
                console.log("■■■■ DIFFERENT HASH ■■■■");
            }
            console.log("prevHash:", blockInfo.previous_block);
            // console.log('Latest Block information:', blockInfo);
            console.log("last height:", blockInfo.height);
            console.log("calcHash:", blockInfo.hash);
            lastHash = blockInfo.hash;
            lastHeight = currentHeight;
        } else {
            console.log("skip height:", blockInfo.height);
        }
    } else {
        const error = await response.text();
        console.error('Error fetching block information:', error);
    }
} catch (error) {
    console.error('Error:', error);
}
};

// Function to fetch block details for the specified height
const fetchBlockDetails = async (height) => {
    try {

```

```

    const response = await fetch(`https://cardano-
preprod.blockfrost.io/api/v0/blocks/${height}`, {
      method: 'GET',
      headers: {
        'project_id': apiKey
      }
    });

    if (response.ok) {
      const blockDetails = await response.json();
      if (lastHash !== blockDetails.previous_block) {
        console.log("■■■■■ DIFFERENT HASH ■■■■■");
      }
      console.log("prevHash:", blockDetails.previous_block);
      console.log("fetch height:", blockDetails.height);
      console.log("calcHash:", blockDetails.hash);
      lastHash = blockDetails.hash;
    } else {
      const error = await response.text();
      console.error(`Error fetching block information for height
${height}:`, error);
    }
  } catch (error) {
    console.error(`Error fetching block information for height
${height}:`, error);
  }
};

// Execute every 15,000 milliseconds (15 seconds)
setInterval(fetchBlockInfo, 15000);
fetchBlockInfo()

```

Execution result

```
[ec2-user@ip-172-31-19-190 node-apps]$ node block_fetch.mjs
■■■■ DIFFERENT HASH ■■■■
prevHash: 65a77e36ae3865ffd3cc5a43fc16eee85f666afd7b9f6d0fee65e8789c3f40b3
last height: 2671842
calcHash: 47a83a16600e678e53c66f35e456d3782613d927b0fd53a81a6ebbeea8cd59df
prevHash: 47a83a16600e678e53c66f35e456d3782613d927b0fd53a81a6ebbeea8cd59df
last height: 2671843
calcHash: 572ece3260af7d447355f0f043c9848dbc4c3819041b8d8a948df18669028c5f
skip height: 2671843
prevHash: 572ece3260af7d447355f0f043c9848dbc4c3819041b8d8a948df18669028c5f
fetch height: 2671844
calcHash: f829d8800a29449137893982fbc3d5806ac5a7a58d80333f0a3b5a30025cc1b6
prevHash: f829d8800a29449137893982fbc3d5806ac5a7a58d80333f0a3b5a30025cc1b6
last height: 2671845
calcHash: 201830c887f88ed10fa22abfcaa0085012477b33a606c8540de9e857b99d472f
skip height: 2671845
skip height: 2671845
skip height: 2671845
prevHash: 201830c887f88ed10fa22abfcaa0085012477b33a606c8540de9e857b99d472f
fetch height: 2671846
calcHash: c779d4adcc8cd21afd6b0de8b9c2c1a618accb81a3ba06214840e42d9518eaf
prevHash: c779d4adcc8cd21afd6b0de8b9c2c1a618accb81a3ba06214840e42d9518eaf
last height: 2671847
calcHash: 2cdf1826ff825e0f3bc7567fda4e8d9e0b1f0a5fd14f6bb4c0fb848600fa2d0c
prevHash: 2cdf1826ff825e0f3bc7567fda4e8d9e0b1f0a5fd14f6bb4c0fb848600fa2d0c
fetch height: 2671848
calcHash: 212e565e8e47c2ee2bb2fab8979d19fdd1d23123eda045417e7b9a754ccffd2b
prevHash: 212e565e8e47c2ee2bb2fab8979d19fdd1d23123eda045417e7b9a754ccffd2b
last height: 2671849
calcHash: 976ed86832c3ae318a78554bcf98357bbab15b94d9d32ac8072f49cce37648a6
prevHash: 976ed86832c3ae318a78554bcf98357bbab15b94d9d32ac8072f49cce37648a6
last height: 2671850
calcHash: 6e26f2d2c17b16fe3f44a3deaa260c2c68c2897385c0f6953a2d1d96be302fee
skip height: 2671850
skip height: 2671850
skip height: 2671850
skip height: 2671850
prevHash: 6e26f2d2c17b16fe3f44a3deaa260c2c68c2897385c0f6953a2d1d96be302fee
last height: 2671851
calcHash: da0c3ea444d6897873421e0b26e91c26fe7f2e2a3d28f154b293bbff03fa9f81
prevHash: da0c3ea444d6897873421e0b26e91c26fe7f2e2a3d28f154b293bbff03fa9f81
last height: 2671852
```

8.3.2 Research on Rollback Process

Using the Cardano Blockchain to Perform Repeated Rollback Processes

Prototype : Source Code (block_rollback.mjs)

```
import fetch from 'node-fetch';
import dotenv from "dotenv";
dotenv.config()

const apiKey = process.env.PROJECT_ID; // Blockfrost API key
let lastHeight = 0;
let lastHash = "";
let pastHashes = [];

// Function to fetch block information from the specified endpoint
const fetchBlock = async (endpoint) => {
  try {
    const response = await fetch(endpoint, {
      method: 'GET',
      headers: {
        'project_id': apiKey
      }
    });
  } catch {
    console.error('Error fetching block information from endpoint');
    return null;
  }

  if (response.ok) {
    return await response.json();
  } else {
    const error = await response.text();
    console.error(`Error fetching block information from ${endpoint}:`,
    error);
    return null;
  }
}
```

```

    }
  } catch (error) {
    console.error(`Error fetching block information from ${endpoint}:`,
error);
    return null;
  }
};

// Fetch the latest block information and update the state
const fetchBlockInfo = async () => {
  const blockInfo = await fetchBlock('https://cardano-
preprod.blockfrost.io/api/v0/blocks/latest');
  if (!blockInfo) return;

  const currentHeight = blockInfo.height;

  if (lastHeight !== currentHeight) {
    if (lastHeight > 0 && currentHeight > lastHeight + 1) {
      // Fetch information for missing blocks
      for (let h = lastHeight + 1; h < currentHeight; h++) {
        await fetchBlockDetails(h);
      }
    }

    if (lastHash !== blockInfo.previous_block) {
      console.log("■■■■■ DIFFERENT HASH ■■■■■");
      await reconcileBlocks(lastHeight);
    }

    updateState(blockInfo);
  } else {
    console.log("skip:", blockInfo.height);
  }
};

```



```

// Fetch the details of the block at the specified height
const fetchBlockDetails = async (height) => {
  const blockDetails = await fetchBlock(`https://cardano-
preprod.blockfrost.io/api/v0/blocks/${height}`);
  if (!blockDetails) return;

  if (lastHash !== blockDetails.previous_block) {
    console.log("■■■■ DIFFERENT HASH ■■■■");
    await reconcileBlocks(height - 1);
  }
  updateState(blockDetails);
};

// Function to reconcile block information
const reconcileBlocks = async (startHeight) => {
  console.log("Reconciling blocks...");
  let currentHeight = startHeight;

  while (currentHeight > 0) {
    const blockDetails = await fetchBlock(`https://cardano-
preprod.blockfrost.io/api/v0/blocks/${currentHeight}`);
    if (!blockDetails) break;

    updateState(blockDetails);
    const pastBlock = pastHashes.find(p => p.height === blockDetails.height);

    if (pastBlock && pastBlock.hash === blockDetails.hash) {
      console.log(`Match found at height ${currentHeight}`);
      lastHeight = currentHeight;
      lastHash = blockDetails.hash;
      break;
    } else {

```

```
        console.log(`No match found at height ${currentHeight}, fetching
previous block`);
        currentHeight--;
    }
}
console.log("Reconciling blocks complete");
};

// Update block information and update the state
const updateState = (blockInfo) => {
    console.log("prevHash:", blockInfo.previous_block);
    // console.log('Block information:', blockInfo);
    console.log("height:", blockInfo.height);
    console.log("calcHash:", blockInfo.hash);

    lastHash = blockInfo.hash;
    lastHeight = blockInfo.height;

    // Add to the list of previous hash values
    pastHashes.push({ height: blockInfo.height, hash: blockInfo.hash });
    if (pastHashes.length > 100) { // 保持するハッシュ値の数を制限 // Limit the
number of stored hash values
        pastHashes.shift();
    }
};

setInterval(fetchBlockInfo, 15000); // Execute every 15,000 milliseconds (15
seconds)
fetchBlockInfo();
```

Execution result

■■■■ DIFFERENT HASH ■■■■

Reconciling blocks...

Reconciling blocks complete

prevHash: 66d7fe20c01d11469c3604014a5c9bddade9a63f992e86daf28d1b70b850fee6

height: 2685246

calcHash: f27ba4cc3a4c093ce40edfa01dccd7ee384ff1b6f0775bd15e303fa75799f98d

skip: 2685246

prevHash: f27ba4cc3a4c093ce40edfa01dccd7ee384ff1b6f0775bd15e303fa75799f98d

height: 2685247

calcHash: e4f9d5f8a93bb6d96b53e4e69817bfcca73c6b48157ab713219e7cf21384c8d4

skip: 2685247

skip: 2685247

skip: 2685247

skip: 2685247

skip: 2685247

skip: 2685247

skip: 2685247

prevHash: e4f9d5f8a93bb6d96b53e4e69817bfcca73c6b48157ab713219e7cf21384c8d4

height: 2685248

calcHash: 84d7bee383a01e5e17ab36a259cb66349b0f1f06433f18ad948423ffb99831cb

skip: 2685248

skip: 2685248

skip: 2685248

prevHash: 84d7bee383a01e5e17ab36a259cb66349b0f1f06433f18ad948423ffb99831cb

height: 2685249

calcHash: eb1facad2359aa03dc060e99103a737e67b833c7e46fe86bebc7912f395e7322

skip: 2685249

■■■ DIFFERENT HASH ■■■

Reconciling blocks...

prevHash: 84d7bee383a01e5e17ab36a259cb66349b0f1f06433f18ad948423ffb99831cb

height: 2685249

calcHash: 882124d0ad418410a608cfacf94547f7d8d5510b7346ff306a1497d39ad32ee2

No match found at height 2685249, fetching previous block

prevHash: e4f9d5f8a93bb6d96b53e4e69817bfcca73c6b48157ab713219e7cf21384c8d4

height: 2685248

calcHash: 84d7bee383a01e5e17ab36a259cb66349b0f1f06433f18ad948423ffb99831cb

Match found at height 2685248

Reconciling blocks complete

prevHash: 882124d0ad418410a608cfacf94547f7d8d5510b7346ff306a1497d39ad32ee2

height: 2685250

calcHash: 717db5d56eb6c9076545ec386f4f01564e25a7bc0d465e0c340015ad457642fd

skip: 2685250

skip: 2685250

skip: 2685250

skip: 2685250

8.3.3 Research on Transaction Root

Investigating Transaction Roots Using the Cardano Blockchain

Prototype : Source Code (block_merkle.js)

```
const axios = require('axios');
const blake = require('blakejs');
require('dotenv').config();

// Fetch transactions for the specified block height from the Blockfrost
API
async function fetchBlockTransactions(height, apiKey) {
  try {
    const response = await axios.get(`https://cardano-
preprod.blockfrost.io/api/v0/blocks/${height}/txs`, {
      headers: { 'project_id': apiKey }
    });
    return response.data;
  } catch (error) {
    console.error('Error fetching block transactions:', error);
    return [];
  }
}

// Generate BLAKE2b hashes from the array of transactions
function getTransactionHashes(transactions) {
  return transactions.map(tx => blake.blake2bHex(tx, null, 32));
}

// Concatenate two hashes and compute the BLAKE2b hash
function hashConcat(left, right) {
  return blake.blake2bHex(left + right, null, 32);
}
```

```

}

// Build a Merkle tree
function buildMerkleTree(hashes) {
  let tree = [hashes];

  while (tree[tree.length - 1].length > 1) {
    let currentLevel = tree[tree.length - 1];
    let nextLevel = [];

    for (let i = 0; i < currentLevel.length; i += 2) {
      const left = currentLevel[i];
      const right = i + 1 < currentLevel.length ? currentLevel[i +
1] : left;
      nextLevel.push(hashConcat(left, right));
    }
    tree.push(nextLevel);
  }
  return tree;
}

(async () => {
  const apiKey = process.env.PROJECT_ID; // Retrieve the API key from
environment variables
  const height = 2528206; // Specify the block height

  // Retrieve transactions within the block
  const transactions = await fetchBlockTransactions(height, apiKey);
  if (transactions.length === 0) {
    console.error('No transactions found');
    return;
  }

  console.log('Transaction IDs:', transactions);

```

```
// Generate transaction hashes and build a Merkle tree
const transactionHashes = getTransactionHashes(transactions);
const tree = buildMerkleTree(transactionHashes);

// Display each level of the Merkle tree
console.log('Merkle Tree:');
tree.forEach((level, index) => {
    console.log(`Level ${index}:`, level);
});

// Display the Merkle root (the topmost hash)
const root = tree[tree.length - 1][0];
console.log('Merkle Root:', root);
})();
```

Execution result

```
[ec2-user@ip-172-31-19-190 node-apps]$node block_merkle.js
Transaction IDs: [
  '79d415fe3ce94447edec54de6496239ad08327280c091bcb26a33e17172e10ec',
  'aa92a3df4bbfacf22108b66f2d6a245002ae8501b51240fb1df8bbab9a759e29'
]
Merkle Tree:
Level 0: [
  'db6cad6ca79e25d75fafd14912936f3d53b9571d499439eca8c69938efa45a1d',
  '5bdb3a75f12ffb47d37750d91173917caf2db251b7fe1d5c71cd5f1eccf406a8'
]
Level 1: [ '701270a92ca73abaa563a7279720f9684f1d28c36fa333a2a6b0c030c44bd233' ]
Merkle Root: 701270a92ca73abaa563a7279720f9684f1d28c36fa333a2a6b0c030c44bd233
```

Transaction IDs:

['79d415fe3ce94447edec54de6496239ad08327280c091bcb26a33e17172e10ec',
'aa92a3df4bbfacf22108b66f2d6a245002ae8501b51240fb1df8bbab9a759e29']

Merkle Tree:

Level 0:

['db6cad6ca79e25d75fafd14912936f3d53b9571d499439eca8c69938efa45a1d',
'5bdb3a75f12ffb47d37750d91173917caf2db251b7fe1d5c71cd5f1eccf406a8']

Level 1:

['701270a92ca73abaa563a7279720f9684f1d28c36fa333a2a6b0c030c44bd233']

Merkle Root:

701270a92ca73abaa563a7279720f9684f1d28c36fa333a2a6b0c030c44bd233