Dogecoin Web Wallet - Technical Documentation

I) Dogecoin address generation

What: functions that generate a new keypair (WIF) and derive DOGE/BTC/FLO/LTC addresses from the same private key.

Functions & code :

`generateNewID()` (helper)

```
// generates a new flo ID and returns private-key, public-key and floID
const generateNewID = (dogeCrypto.generateNewID = function () {
  var key = new Bitcoin.ECKey(false);
  key.setCompressed(true);
  return {
    floID: key.getBitcoinAddress(),
    pubKey: key.getPubKeyHex(),
    privKey: key.getBitcoinWalletImportFormat(),
  };
});
```

Source: `dogeCrypto.js`.

How used: used internally by `generateMultiChain()` to create a new WIF when no WIF is provided.

`generateMultiChain(inputWif)` — main multi-chain generator

```
dogeCrypto.generateMultiChain = function (inputWif) {
  try {
      const origBitjsPub = bitjs.pub;
      const origBitjsPriv = bitjs.priv;
      const origBitjsCompressed = bitjs.compressed;
      const origCoinJsCompressed = coinjs.compressed;

      bitjs.compressed = true;
      coinjs.compressed = true;

      const versions = {
          DOGE: { pub: 0x1e, priv: 0x9e },
          BTC: { pub: 0x00, priv: 0x80 },
          FLO: { pub: 0x23, priv: 0xa3 },
          LTC: { pub: 0x30, priv: 0xb0 },
      };

      let privKeyHex;
      let compressed = true;

      if (typeof inputWif === "string" && inputWif.length > 0) {
```

```
    const decode = Bitcoin.Base58.decode(inputWif);
    const keyWithVersion = decode.slice(0, decode.length - 4);
    let key = keyWithVersion.slice(1);

    if (key.length >= 33 && key[key.length - 1] === 0x01) {
        key = key.slice(0, key.length - 1);
        compressed = true;
    } else {
        compressed = false;
    }

    privKeyHex = Crypto.util.bytesToHex(key);
} else {
    const newKey = generateNewID();
    const decode = Bitcoin.Base58.decode(newKey.privKey);
    const keyWithVersion = decode.slice(0, decode.length - 4);
    let key = keyWithVersion.slice(1);

    if (key.length >= 33 && key[key.length - 1] === 0x01) {
        key = key.slice(0, key.length - 1);
    }

    privKeyHex = Crypto.util.bytesToHex(key);
}

bitjs.compressed = compressed;
coinjs.compressed = compressed;

// Generate public key
const pubKey = bitjs.newPubkey(privKeyHex);

const result = {
    DOGE: { address: "", privateKey: "" },
    BTC: { address: "", privateKey: "" },
    FLO: { address: "", privateKey: "" },
    LTC: { address: "", privateKey: "" },
};

// For DOGE
bitjs.pub = versions.DOGE.pub;
bitjs.priv = versions.DOGE.priv;
result.DOGE.address = bitjs.pubkey2address(pubKey);
result.DOGE.privateKey = bitjs.privkey2wif(privKeyHex);

// For BTC
bitjs.pub = versions.BTC.pub;
bitjs.priv = versions.BTC.priv;
result.BTC.address = coinjs.bech32Address(pubKey).address;
result.BTC.privateKey = bitjs.privkey2wif(privKeyHex);

// For FLO
bitjs.pub = versions.FLO.pub;
bitjs.priv = versions.FLO.priv;
result.FLO.address = bitjs.pubkey2address(pubKey);
result.FLO.privateKey = bitjs.privkey2wif(privKeyHex);

// For LTC
bitjs.pub = versions.LTC.pub;
bitjs.priv = versions.LTC.priv;
result.LTC.address = bitjs.pubkey2address(pubKey);
result.LTC.privateKey = bitjs.privkey2wif(privKeyHex);
```

```
      bitjs.pub = origBitjsPub;
      bitjs.priv = origBitjsPriv;
      bitjs.compressed = origBitjsCompressed;
      coinjs.compressed = origCoinJsCompressed;

      return result;
  } catch (error) {
      console.error("Error in generateMultiChain:", error);
      throw error;
  }
};
```

Source: `dogeCrypto.js`.

How it works:

- If `inputWif` given, decode it for the raw private key; otherwise create a new private key.
- Build pubkey from privKeyHex, then re-encode that pubkey with different version bytes to produce DOGE/BTC/FLO/LTC addresses and corresponding WIFs.

`translateAddress(address)` — address-to-address translation

```
dogeCrypto.translateAddress = function (address) {
  try {
    let sourceChain = null;

    if (address.startsWith("bc1")) {
      sourceChain = "BTC";
    } else if (address.startsWith("D")) {
      sourceChain = "DOGE";
    } else if (address.startsWith("F")) {
      sourceChain = "FLO";
    } else if (address.startsWith("L")) {
      sourceChain = "LTC";
    } else {
      throw new Error("Unsupported address format");
    }

    let decoded, hash160;

    if (sourceChain === "BTC") {
      decoded = coinjs.bech32_decode(address);
      if (!decoded) throw new Error("Invalid bech32 address");

      // For segwit addresses, convert from 5-bit to 8-bit
      const data = coinjs.bech32_convert(decoded.data.slice(1), 5, 8,
false);
      hash160 = Crypto.util.bytesToHex(data);
    } else {
      // Handle DOGE and FLO addresses (Base58)
      const decodedBytes = Bitcoin.Base58.decode(address);
      if (!decodedBytes || decodedBytes.length < 25)
        throw new Error("Invalid address");
```

```
      // Remove version byte (first byte) and checksum (last 4 bytes)
      const bytes = decodedBytes.slice(1, decodedBytes.length - 4);
      hash160 = Crypto.util.bytesToHex(bytes);
    }

    if (!hash160) throw new Error("Could not extract hash160 from
address");

    const versions = {
      DOGE: 0x1e,
      FLO: 0x23,
      BTC: 0x00,
      LTC: 0x30,
    };

    const result = {};

    // Generate address for DOGE
    const dogeBytes = Crypto.util.hexToBytes(hash160);
    dogeBytes.unshift(versions.DOGE);
    const dogeChecksum = Crypto.SHA256(
      Crypto.SHA256(dogeBytes, { asBytes: true }),
      { asBytes: true }
    ).slice(0, 4);
    result.DOGE = Bitcoin.Base58.encode(dogeBytes.concat(dogeChecksum));

    // Generate address for FLO
    const floBytes = Crypto.util.hexToBytes(hash160);
    floBytes.unshift(versions.FLO);
    const floChecksum = Crypto.SHA256(
      Crypto.SHA256(floBytes, { asBytes: true }),
      { asBytes: true }
    ).slice(0, 4);
    result.FLO = Bitcoin.Base58.encode(floBytes.concat(floChecksum));

    // Generate address for BTC
    try {
      const words = coinjs.bech32_convert(
        Crypto.util.hexToBytes(hash160),
        8,
        5,
        true
      );
      result.BTC = coinjs.bech32_encode("bc", [0].concat(words));
    } catch (e) {
      console.log("Could not generate segwit address:", e);
    }

    // Generate address for LTC
    const ltcBytes = Crypto.util.hexToBytes(hash160);
    ltcBytes.unshift(versions.LTC);
    const ltcChecksum = Crypto.SHA256(
      Crypto.SHA256(ltcBytes, { asBytes: true }),
      { asBytes: true }
    ).slice(0, 4);
    result.LTC = Bitcoin.Base58.encode(ltcBytes.concat(ltcChecksum));

    return result;
  } catch (err) {
    console.error("Address translation error:", err);
    throw new Error("Address translation failed: " + err.message);
```

```
  }
};
```

Source: `dogeCrypto.js`.

How it's invoked in UI: `translateDirectAddress()` wrapper in `index.html` calls `dogeCrypto.translateAddress(address)` and displays the result. See `index.html` below.

`verifyPrivKey(privateKeyWIF, dogeAddress)` and `validateDogeID(address)` (helpers)

```
// verifyPrivKey returns true if WIF derives the given DOGE address
dogeCrypto.verifyPrivKey = function (privateKeyWIF, dogeAddress) {
  if (!privateKeyWIF || !dogeAddress) return false;
  try {
    var derivedAddress =
      dogeCrypto.generateMultiChain(privateKeyWIF).DOGE.address;
    return derivedAddress === dogeAddress;
  } catch (e) {
    console.error("verifyPrivKey error:", e);
    return false;
  }
};

// validateDogeID returns true if version byte matches DOGE prefix
dogeCrypto.validateDogeID = function (dogeID) {
  if (!dogeID) return false;
  try {
    // Decode Base58Check
    let bytes = bitjs.Base58.decode(dogeID);
    if (!bytes || bytes.length < 25) return false;
    let version = bytes[0];

    return version === 0x1e;
  } catch (e) {
    return false;
  }
};
```

Source: `dogeCrypto.js`.

How used: `verifyPrivKey` is used by the send function to verify the provided WIF matches the sender address. `validateDogeID` validates addresses before operations. See `dogeBlockchainAPI.sendDogecoinRPC`.

UI wrappers (from `index.html`)

```
function generateMultiChain() {
  try {
    const result = dogeCrypto.generateMultiChain();
    document.getElementById("multiResult").innerText = JSON.stringify(
```

```
      result,
      null,
      2
    );
  } catch (err) {
    document.getElementById("multiResult").innerText =
      "Error: " + err.message;
  }
}

function translateAddress() {
  const wif = document.getElementById("translateWIF").value.trim();
  try {
    const result = dogeCrypto.generateMultiChain(wif);
    document.getElementById("translateResult").innerText = JSON.stringify(
      result,
      null,
      2
    );
  } catch (err) {
    document.getElementById("translateResult").innerText =
      "Error: " + err.message;
  }
}

function translateDirectAddress() {
  const address =
document.getElementById("addressToTranslate").value.trim();
  try {
    const result = dogeCrypto.translateAddress(address);
    document.getElementById("directTranslateResult").innerText =
      JSON.stringify(result, null, 2);
  } catch (err) {
    document.getElementById("directTranslateResult").innerText =
      "Error: " + err.message;
  }
}
```

Source: `index.html`.

II) Fetching DOGE balance and all Dogecoin transactions (given DOGE / FLO / BTC address)

What: functions that query the REST endpoint and return balance and processed, paginated transaction history (with `sent/received/self` classification).

Functions & code :

`getBalance(addr)`

```
dogeBlockchainAPI.getBalance = function (addr) {
  return new Promise((resolve, reject) => {
    fetch(
      `https://go.getblock.io/getblock_token/api/address/${addr}`
```

```
    )
      .then((response) => {
        if (!response.ok)
          throw new Error(`HTTP error! Status: ${response.status}`);
        return response.json();
      })
      .then((data) => {
        console.log("Balance data:", data);
        if (data && typeof data.balance !== "undefined")
          resolve(parseFloat(data.balance));
        else reject("Balance not found in response");
      })
      .catch((error) => reject(error));
  });
};
```

Source: `dogeBlockchainAPI.js`.

How used in UI: `checkDogeBalance()` in `index.html` calls this and displays the returned balance.

`getDogeTransactions(address, options)` — transaction history + classification & pagination

```
dogeBlockchainAPI.getDogeTransactions = function (address, options = {}) {
  return new Promise((resolve, reject) => {
    console.log(`Fetching transaction history for: ${address}`);

fetch(`https://go.getblock.io/getblocktoken/api/address/${address}?details=
txs`)
      .then((response) => {
        if (!response.ok) {
          if (response.status === 429) {
            throw new Error(
              "API rate limit exceeded. Please try again later."
            );
          }
          throw new Error(`HTTP error! Status: ${response.status}`);
        }
        return response.json();
      })
      .then(async (data) => {
        console.log("Raw API response data:", data);
        const txs = data.txs || [];
        const txids = txs.map((tx) => tx.txid) || [];
        console.log(
          `Found ${txids.length} transactions for address ${address}`
        );
        const limit = options.limit || 10;
        const offset = options.offset || 0;

        const maxTxToProcess = Math.min(10, limit);
        const txsToProcess = txs.slice(offset, offset + maxTxToProcess);

        if (txsToProcess.length === 0) {
          console.log("No transactions to process based on offset/limit");
```

```
  resolve({
    transactions: [],
    total: txs.length,
    offset: offset,
    limit: limit,
  });
  return;
}

console.log(`Processing ${txsToProcess.length} transactions`);

const transactions = txsToProcess;
console.log("Transactions to process:", transactions  );

try {
  const processedTransactions = transactions.map((tx) => {
    const inputs = tx.vin || [];
    const outputs = tx.vout || [];

    // Check if address is sender (in vin)
    const isSender = inputs.some((i) =>
      i.addresses?.includes(address)
    );

    // Check if address is receiver (in vout)
    const isReceiver = outputs.some(
      (o) =>
        (o.addresses && o.addresses.includes(address)) ||
        (o.scriptPubKey?.addresses &&
          o.scriptPubKey.addresses.includes(address))
    );

    let type = "unknown";
    let value = 0;

    if (isSender && isReceiver) {
      type = "self";

      const totalInput = inputs
        .filter((i) => i.addresses?.includes(address))
        .reduce((sum, i) => sum + toDOGE(i.value), 0);

      const totalOutput = outputs
        .filter(
          (o) =>
            (o.addresses && o.addresses.includes(address)) ||
            (o.scriptPubKey?.addresses &&
              o.scriptPubKey.addresses.includes(address))
        )
        .reduce((sum, o) => sum + toDOGE(o.value), 0);

      value = totalOutput - totalInput;
    } else if (isSender) {
      type = "sent";

      const totalInput = inputs
        .filter((i) => i.addresses?.includes(address))
        .reduce((sum, i) => sum + toDOGE(i.value), 0);

      const changeBack = outputs
        .filter(
```

```
            (o) =>
              (o.addresses && o.addresses.includes(address)) ||
              (o.scriptPubKey?.addresses &&
                o.scriptPubKey.addresses.includes(address))
          )
          .reduce((sum, o) => sum + toDOGE(o.value), 0);

        value = -(totalInput - changeBack);
      } else if (isReceiver) {
        type = "received";

        value = outputs
          .filter(
            (o) =>
              (o.addresses && o.addresses.includes(address)) ||
              (o.scriptPubKey?.addresses &&
                o.scriptPubKey.addresses.includes(address))
          )
          .reduce((sum, o) => sum + toDOGE(o.value), 0);
      }

      console.log(`Transaction ${tx.txid} time data:`, {
        blockTime: tx.blocktime,
        blockheight: tx.blockheight,
        time: tx.time,
      });

      const timestamp =
        tx.time ||
        tx.blockTime ||
        (tx.confirmations
          ? Math.floor(Date.now() / 1000) - tx.confirmations * 600
          : Math.floor(Date.now() / 1000));

      return {
        txid: tx.txid,
        type,
        value: value.toFixed(8),
        time: timestamp,
        blockHeight:  tx.blockheight,
        formattedTime: new Date(timestamp * 1000).toLocaleString(),
        confirmations: tx.confirmations || 0,
        rawTx: tx.hex,
      };
    });

    resolve({
      transactions: processedTransactions,
      total: txids.length,
      offset: offset,
      limit: limit,
    });
  } catch (error) {
    console.error("Error processing transactions:", error);
    reject(error);
  }
})
.catch((error) => {
  console.error("API Error:", error);
  reject(error);
});
```

```
  });
};
```

Source: `dogeBlockchainAPI.js`.

How it's invoked in UI: `fetchTransactionsWithPagination()` in `index.html` calls `dogeBlockchainAPI.getDogeTransactions(currentTxAddress, { limit: txPerPage, offset: currentTxOffset })` and then renders rows, sets pagination controls, and classifies rows into Received / Sent / Self based on `value` sign.

Helper: `toDOGE(val)` and `getUTXOs(addr)` (used for UTXO listing & value parsing)

```
function toDOGE(val) {
  if (typeof val === "string" && val.includes("DOGE")) {
    return parseFloat(val.replace("DOGE", "").trim());
  }
  const num = parseFloat(val || "0");
  return isNaN(num) ? 0 : num;
}

// Helper function to get UTXOs for an address
const getUTXOs = async (addr) => {
  const url =
`https://go.getblock.io/getblocktoken/api/address/${addr}?details=txs`;
  const res = await fetch(url);
  const data = await res.json();
  if (!data.txs) throw new Error("No transactions found for address");

  const utxos = [];
  data.txs.forEach((tx) => {
    tx.vout.forEach((vout) => {
      const addresses =
        vout.addresses ||
        (vout.scriptPubKey ? vout.scriptPubKey.addresses : []);
      if (
        !vout.spent &&
        vout.scriptPubKey &&
        vout.scriptPubKey.hex &&
        addresses &&
        addresses.some((a) => a.toLowerCase() === addr.toLowerCase())
      ) {
        utxos.push({
          txid: tx.txid,
          vout: vout.n,
          value: parseFloat(vout.value),
          scriptPubKey: vout.scriptPubKey.hex,
        });
      }
    });
  });
  return utxos;
};
```

Source: `dogeBlockchainAPI.js`.

How used: `getUTXOs` is used by send flow to gather spendable UTXOs for a sender address.

III) Sending DOGE

What: create raw transaction from UTXOs, sign it using the provided WIF, and broadcast via GetBlock JSON-RPC (`createrawtransaction` → `signrawtransaction` → `sendrawtransaction`).

Functions & code :

`sendDogecoinRPC(senderAddr, receiverAddr, sendAmt, privKey)` - full send flow

```
dogeBlockchainAPI.sendDogecoinRPC = function (
  senderAddr,
  receiverAddr,
  sendAmt,
  privKey
) {
  return new Promise((resolve, reject) => {
    if (!dogeCrypto.validateDogeID(senderAddr, true))
      return reject(`Invalid sender address: ${senderAddr}`);
    if (!dogeCrypto.validateDogeID(receiverAddr))
      return reject(`Invalid receiver address: ${receiverAddr}`);
    if (typeof sendAmt !== "number" || sendAmt <= 0)
      return reject(`Invalid send amount: ${sendAmt}`);
    if (privKey.length < 1 || !dogeCrypto.verifyPrivKey(privKey,
senderAddr))
      return reject("Invalid Private key!");

    const fee = DEFAULT.fee;
    const apiToken = getblock-token;
    const rpcEndpoint = `https://go.getblock.io/${apiToken}/`;

    async function rpc(method, params = []) {
      const res = await fetch(rpcEndpoint, {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ jsonrpc: "2.0", id: "1", method, params }),
      });
      const text = await res.text();
      try {
        const data = JSON.parse(text);
        if (data.error) throw new Error(JSON.stringify(data.error));
        return data.result;
      } catch (err) {
        console.error("Raw RPC response:\n", text);
        throw new Error("Failed to parse JSON-RPC response");
      }
    }

    // Get UTXOs for the address
    getUTXOs(senderAddr)
      .then(async (utxos) => {
        if (utxos.length === 0) return reject("No valid UTXOs found");
```

```
        const utxoTotal = utxos.reduce((sum, utxo) => sum + utxo.value, 0);

        if (utxoTotal < sendAmt + fee)
          return reject(
            `Insufficient funds: ${utxoTotal} < ${sendAmt + fee}`
          );

        const inputs = utxos.map((utxo) => ({
          txid: utxo.txid,
          vout: utxo.vout,
        }));

        // Calculate change amount
        const change = utxoTotal - sendAmt - fee;

        const outputs = {
          [senderAddr]: Number(change.toFixed(8)),
          [receiverAddr]: Number(sendAmt.toFixed(8)),
        };

        try {
          // Create raw transaction
          const rawTx = await rpc("createrawtransaction", [inputs,
outputs]);

          // Sign raw transaction
          const signedTx = await rpc("signrawtransaction", [
            rawTx,
            [
              {
                txid: utxos[0].txid,
                vout: utxos[0].vout,
                scriptPubKey: utxos[0].scriptPubKey,
                amount: utxos[0].value.toFixed(8),
              },
            ],
            [privKey],
          ]);

          if (!signedTx.complete) {
            return reject(
              `Failed to sign transaction:
${JSON.stringify(signedTx.errors)}`
            );
          }

          // Send raw transaction
          const txid = await rpc("sendrawtransaction", [signedTx.hex]);

          resolve(txid);
        } catch (error) {
          reject(error);
        }
      })
      .catch((error) => reject(error));
  });
};
```

Source: `dogeBlockchainAPI.js`.

Important notes about this code:

- It validates addresses and that the provided WIF corresponds to the sender address (`verifyPrivKey`).
- It uses a hardcoded `fee` (DEFAULT.fee = 0.09) — you may want to compute fees dynamically.

UI wrapper: `sendDogeRPC()` (index.html)

```
function sendDogeRPC() {
  const senderAddress = document
    .getElementById("senderAddress")
    .value.trim();
  const privateKey = document.getElementById("privateKey").value.trim();
  const receiverAddress = document
    .getElementById("receiverAddress")
    .value.trim();
  const sendAmount = parseFloat(
    document.getElementById("sendAmount").value
  );

  if (
    !senderAddress ||
    !privateKey ||
    !receiverAddress ||
    isNaN(sendAmount) ||
    sendAmount <= 0
  ) {
    document.getElementById("sendResult").innerText =
      "Error: Please fill in all fields with valid values";
    return;
  }

  document.getElementById("sendResult").innerText =
    "Processing transaction...";

  dogeBlockchainAPI
    .sendDogecoinRPC(
      senderAddress,
      receiverAddress,
      sendAmount,
      privateKey
    )
    .then((txid) => {
      document.getElementById(
        "sendResult"
      ).innerHTML = `<p>Transaction successful!</p>
        <p>Transaction ID: <a
href="https://blockchair.com/dogecoin/transaction/${txid}"
target="_blank">${txid}</a></p>`;
    })
    .catch((error) => {
      document.getElementById("sendResult").innerText =
        "Transaction failed: " + (error.message || error);
      console.error("Transaction error:", error);
    });
```

```
}
```

Source: `index.html`.
