Signing a Transaction

Suggest Edits

The Magic Eden Wallet support the Sats Connect method ofsigning a transaction . This requires a base64 encoded PSBT to execute. Let's walk through each step of this process.

Let's take a look at a basic example, where we want to send an arbitrary amount of BTC to another wallet. The top-level function call should look something like this:

JavaScript async function signWalletTransaction() { const utxos = await fetchUTXO(nativeSegwitAddress); if (!utxos) { alert('No UTXOs found! Please deposit some funds to your wallet.'); return; } const selectedUTXO = await selectUTXO(10000, utxos); const scriptPubKey = await fetchScriptPubKey(selectedUTXO.txid, selectedUTXO.vout); const psbt = await createPSBT({ utxo: selectedUTXO, recipientAddress: RECIPIENT_ADDRESS, changeAddress: nativeSegwitAddress, amountToSend: 1000, scriptPubKey, });

try { await signTransaction({ payload: { network: { type: BitcoinNetworkType.Mainnet, }, psbtBase64: psbt, broadcast: true, message: "tip the author! Don't worry this will not be broadcasted.", inputsToSign: [{ address: nativeSegwitAddress!, signingIndexes: [0], },], }, onFinish: (response) => { const psbResponse = Psbt.fromBase64(response.psbtBase64); psbResponse.finalizeAllInputs(); const signedTx = psbResponse.extractTransaction(); const txHex = signedTx.toHex(); }, onCancel: () => { alert('Request canceled'); }, }); } catch (err) { console.error(err); } } There's a lot going on here, so let's break it down.

Understanding UTXOs:

In the world of Bitcoin, the concept of a "balance" operates differently compared to traditional bank accounts or even some other blockchain networks. Rather than representing your funds as a single total amount, your Bitcoin balance is actually the sum of all values from Unspent Transaction Outputs (UTXOs). UTXOs are essentially fragments of Bitcoin left over from previous transactions that you have received but not yet spent.

When someone sends you Bitcoin, the amount is recorded in the blockchain as a UTXO, earmarked for your address. Your wallet then aggregates all UTXOs associated with your address to display your total balance. It's similar to having a wallet full of various bills and coins; each piece has value, and your total wealth is the sum of these individual pieces.

To initiate a Bitcoin transaction, your wallet selects enough UTXOs to cover the amount you wish to send, plus the transaction fee. These selected UTXOs are then consumed and effectively transformed: the amount sent to the recipient becomes a new UTXO linked to their address, and any change not sent to the recipient (the remainder after subtracting the amount sent and the transaction fee) returns to you as a new UTXO, ready to be spent in future transactions.

In this particular example, we are just going with the first UTXO that fits the bill. Actual production implementations should expand on this. An easy way to fetch all UTXO's of an address is utilizingmempool's API

JavaScript /* ** Fetches UTXOs from the mempool.space API for a specified address. * * @param {string} address The Bitcoin address to fetch UTXOs for. * @returns {Promise} A promise that resolves with an array of UTXOs for the given address. * @throws {Error} If the request to the mempool.space API fails. Most likely due to not found (no balance) or invalid address.* / export const fetchUTXO = async (address: string): Promise=> { const response = await fetch(https://mempool.space/api/address{address}/utxo); if (!response.ok) { throw new Error('Error fetching UTXO from mempool.space API.'); } return response.json(); }; Feel free to reference thedemo at any point to view the full source code if the snippets feel sparse.

ScriptPubKeys and WitnessUTXOs

In this example, we are looking at a transaction payment. That makes use of our segwit address instead of our ordinal address. In the ME wallet, your main BTC address for sending bitcoin is a native segwit address.SegWit (or Segregated Witness) was introduced to make Bitcoin transactions more efficient and secure

In the context of creating PSBTs, ScriptPubKey and WitnessUTXO are essential for SegWit transactions because they ensure that bitcoins are securely and efficiently sent across the Bitcoin network. ScriptPubKey ensures that only the intended recipient can spend the bitcoins, while WitnessUTXO makes transactions more space-efficient, allowing for faster processing and lower fees.

You can find the given scriptPubKey for a UTXO transaction id from the mempool APIs as well:

JavaScript /* ** Fetches the scriptPubKey for a given UTXO transactionId from the mempool.space API. * This is necessary for creating a PSBT for a segwit transaction. * * @param {string} txId The transaction ID of the UTXO. * @param {number} vout The vout index of the UTXO. * @returns {Promise} A promise that resolves with the hex representation of the scriptPubKey. * @throws {Error} If the request to the mempool.space API fails, or if the scriptPubKey is not found for the* *given vout.* / export const fetchScriptPubKey = async (txId: string, vout: number): Promise=> { const response = await fetch(https://mempool.space/api/tx{txId}); if (!response.ok) { throw new Error('Error fetching transaction details from mempool.space API.'); } const transaction = await response.json(); if (transaction.vout && transaction.vout.length > vout && transaction.vout[vout].scriptpubkey) { return transaction.vout[vout].scriptpubkey; } else { throw new Error('scriptPubKey not found for the given vout.'); } };

Creating PSBTs

Now for the fun part! We've put together all the pieces, now let's create a PSBT that we can actually execute. PSBTs are a flexible, standardized format for Bitcoin transactions that can be partially signed and completed by multiple parties.

This example walks through creating a PSBT for Segwit transactions, but taproot transactions are similar and you can follow the documentation fromsats-connect if need be. This example process involves sending Bitcoin from one address to another and returning any leftover balance from the UTXO back to the sender.

JavaScript import { UTXO } from './fetchUTXO'; import { Psbt, networks } from 'bitcoinjs-lib';

interface SegwitProps { utxo: UTXO; recipientAddress: string; changeAddress: string; amountToSend: number; scriptPubKey?: string; senderPubKey?: string; }

/* ** A basic implementation of creating a PSBT for a segwit transaction. This consists of * sending btc from one address to another, returning the leftover balance from the UTXO to the sender. * For a real-world application, you would want to implement more checks and error handling. * * @param {SegwitProps} props The properties needed to create a PSBT. * @returns {Promise} A promise that resolves with the base64 representation of the PSBT.* / export const createPSBT = async ({ utxo, recipientAddress, changeAddress, amountToSend, scriptPubKey, }: SegwitProps) => { const psbt = new Psbt({ network: networks.bitcoin });

```
// change to return to sender minus the amount to send and the transaction fee (500 sats for this example)
const changeValue = utxo.value! - amountToSend - 500;

psbt.addInput({
    hash: utxo?.txid!,
    index: utxo?.vout!,
    witnessUtxo: {
        script: Buffer.from(scriptPubKey!, 'hex'),
        value: utxo.value!,
    },
});

psbt.addOutput({
    address: recipientAddress,
    value: amountToSend,
});

// change from the UTXO needs to be returned to the sender
psbt.addOutput({
    address: changeAddress,
    value: changeValue,
});

return psbt.toBase64();
```

}; This formed PSBT can be decoded and sent back to the original code snippet to be executed in our SignTransaction function. Updatedabout 1 month ago

* Table of Contents * * Understanding UTXOs: * * ScriptPubKeys and WitnessUTXOs * * Creating PSBTs