

Forum:[Hack Forums](#)

Thread: [Need Pseudo-Signature PSBT Script Help Wanted!](#)

keiwn2 - August 31, 2024, 10:01 am

I am currently working on a project that involves creating a Partially Signed Bitcoin Transaction (PSBT) with a pseudo-signature. I understand that this type of transaction is non-standard and won't be valid under typical Bitcoin network rules, but it is necessary for the specific requirements of my project.

I am reaching out to see if anyone has experience or resources that could help in generating a PSBT with a pseudo-signature. Specifically, I am looking for:

A script or code snippet that can be used to create a PSBT with a pseudo-signature, ideally in a language such as JavaScript (using libraries like bitcoinjs-lib), Python, or any other relevant environment.

Links to tutorials or documentation that provide detailed guidance on creating such a transaction.

Thank you, everyone.

notagh0st - August 31, 2024, 12:34 pm

(Aug 31, 2024 10:01 AM)keiwn2 Wrote: [\[CITE\]](#) I am currently working on a project that involves creating a Partially Signed Bitcoin Transaction (PSBT) with a pseudo-signature. I understand that this type of transaction is non-standard and won't be valid under typical Bitcoin network rules, but it is necessary for the specific requirements of my project.

I am reaching out to see if anyone has experience or resources that could help in generating a PSBT with a pseudo-signature. Specifically, I am looking for:

A script or code snippet that can be used to create a PSBT with a pseudo-signature, ideally in a language such as JavaScript (using libraries like bitcoinjs-lib), Python, or any other relevant environment.

Links to tutorials or documentation that provide detailed guidance on creating such a transaction.

Thank you, everyone.

I did some experimenting with this a while ago but never actually figured it out until just now.

I'm using nodejs because I was having a hard time getting some of the python libraries to work in my environment.

Generate a testnet address:

```
Code const { Psbt, networks, payments } = require('bitcoinjs-lib');
const { ECPairFactory } = require('ecpair');
const axios = require('axios');
const network = networks.testnet;
const tinysecp = require('tiny-secp256k1');
const ECPair = ECPairFactory(tinysecp);
```

```
const keyPair = ECPair.makeRandom({ network });
const { address } = payments.p2wpkh({ pubkey: keyPair.publicKey, network });
```

```
console.log('-----');
console.log('Address:', address);
console.log('Public Key:', keyPair.publicKey);
console.log('Hex Public Key:', keyPair.publicKey.toString('hex'));
console.log('Private Key:', keyPair.privateKey);
console.log('Hex Private Key:', keyPair.privateKey.toString('hex'));
console.log('-----');
```

Use a site like <https://bitcoinfaucet.uo1.net/> to fund the new address on the testnet, wait for the transactions to be confirmed.

Then make sure you see the transaction when viewing:

<https://blockstream.info/testnet/api/address/<address>here>/utxo>

Once you see it, do something like:

```
Code let address = 'tb1qqgut3npe5j4vwhudt2xa7n7uvdn5t65cvecxhla';
let publicKey = '025e12320573a0454fb2e1cf52a7731967c074e31b2195cb5f00e42afd2fd0c5a5';
let privateKey = '99b5c836c102db29e209649d3efb32c965a90a7cc50a4c2bb167be062767be9b';
```

```
let publicKeyBuffer = Buffer.from(publicKey, 'hex');
let privateKeyBuffer = Buffer.from(privateKey, 'hex');
```

```
let keyPair = ECPair.fromPrivateKey(privateKeyBuffer, { network: networks.testnet });
```

```
async function fetchUtxo(address) {
  try {
    const response = await axios.get(`https://blockstream.info/testnet/api/address/${address}/utxo`);
    const utxos = response.data;

    if (utxos.length === 0) {
      throw new Error('No UTXOs found for the address');
    }

    return utxos[0];
  } catch (error) {
    console.error('Error fetching UTXO:', error.message);
    process.exit(1);
  }
}
```

```
async function fetchTransactionHex(txid) {
  try {
    const response = await axios.get(`https://blockstream.info/testnet/api/tx/${txid}/hex`);
    return response.data;
  } catch (error) {
    console.error('Error fetching transaction hex:', error.message);
    process.exit(1);
  }
}
```

```

async function createPsbt() {
  const utxo = await fetchUtxo(address);
  const prevTxHex = await fetchTransactionHex(utxo.txid);

  const psbt = new Psbt({ network });

  psbt.addInput({
    hash: utxo.txid,
    index: utxo.vout,
    nonWitnessUtxo: Buffer.from(prevTxHex, 'hex'),
  });

  psbt.addOutput({
    address: address,
    value: utxo.value - 1000,
  });

  psbt.signInput(0, keyPair);
  psbt.finalizeAllInputs();

  const psbtBase64 = psbt.toBase64();
  console.log('Generated PSBT in Base64:', psbtBase64);
}

```

createPsbt();

This should give you the psbt, then from there you can do something like:

```

Code async function createPseudoSignature(psbtBase64, publicKeyHex) {
  let psbt = Psbt.fromBase64(psbtBase64);

```

```

    let pseudoSig = Buffer.concat([Buffer.alloc(64, 0), Buffer.from([0x01])]); // generate a 64 byte
buffer with 0s and append 0x01 for SIGHASH_ALL

```

```

    let publicKey = Buffer.from(publicKeyHex, 'hex');

```

```

    psbt.data.inputs.forEach(input => {
      if (!input.partialSig) {
        input.partialSig = [];
      }
      input.partialSig.push({
        pubkey: publicKey,
        signature: pseudoSig
      });
    });

```

```

    return psbt.toBase64();
  }

```

All together, something like this:

```

Code // create the address with public/private keypairs
const { Psbt, networks, payments } = require('bitcoinjs-lib');
const { ECPairFactory } = require('ecpair');

```

```
const axios = require('axios');
const network = networks.testnet;
const tinysecp = require('tiny-secp256k1');
const ECPair = ECPairFactory(tinysecp);
const keyPair = ECPair.makeRandom({ network });
const { address } = payments.p2wpkh({ pubkey: keyPair.publicKey, network });
```

```
console.log('-----');
console.log('Address:', address);
console.log('Public Key:', keyPair.publicKey);
console.log('Hex Public Key:', keyPair.publicKey.toString('hex'));
console.log('Private Key:', keyPair.privateKey);
console.log('Hex Private Key:', keyPair.privateKey.toString('hex'));
console.log('-----');
```

// after testnet transaction has been confirmed:

```
let address = '<generated address here>';
let publicKey = '<hex public key>';
let privateKey = '<hex private key>';
```

```
let publicKeyBuffer = Buffer.from(publicKey, 'hex');
let privateKeyBuffer = Buffer.from(privateKey, 'hex');
```

```
let keyPair = ECPair.fromPrivateKey(privateKeyBuffer, { network: networks.testnet });
```

```
async function fetchUtxo(address) {
  try {
    let response = await axios.get(`https://blockstream.info/testnet/api/address/${address}/utxo`);
    let utxos = response.data;

    if (utxos.length === 0) {
      throw new Error('No transactions found');
    }

    return utxos[0];
  } catch (error) {
    console.error('Error fetching transactions:', error.message);
    process.exit(1);
  }
}
```

```
async function fetchTransactionHex(txid) {
  try {
    let response = await axios.get(`https://blockstream.info/testnet/api/tx/${txid}/hex`);
    return response.data;
  } catch (error) {
    console.error('Error fetching transaction hex:', error.message);
    process.exit(1);
  }
}
```

```
async function createPsbt() {
```

```

let utxo = await fetchUtxo(address);
let prevTxHex = await fetchTransactionHex(utxo.txid);

let psbt = new Psbt({ network });

psbt.addInput({
  hash: utxo.txid,
  index: utxo.vout,
  nonWitnessUtxo: Buffer.from(prevTxHex, 'hex'),
});

psbt.addOutput({
  address: address,
  value: utxo.value - 1000,
});

psbt.signInput(0, keyPair);
psbt.finalizeAllInputs();

let psbtBase64 = psbt.toBase64();
return psbtBase64;
}

async function createPseudoSignature(psbtBase64, publicKeyHex) {
  let psbt = Psbt.fromBase64(psbtBase64);

  let pseudoSig = Buffer.concat([Buffer.alloc(64, 0), Buffer.from([0x01])]); // generate a 64 byte
  buffer with 0s and append 0x01 for SIGHASH_ALL

  let publicKey = Buffer.from(publicKeyHex, 'hex');

  psbt.data.inputs.forEach(input => {
    if (!input.partialSig) {
      input.partialSig = [];
    }
    input.partialSig.push({
      pubkey: publicKey,
      signature: pseudoSig
    });
  });

  return psbt.toBase64();
}

(async () => {
  let psbtBase64 = await createPsbt();
  console.log('PSBT in Base64:', psbtBase64);
  let psbtWithPsudoSignature = await createPseudoSignature(psbtBase64, publicKey);
  console.log('PSBT with pseudo signature:', psbtWithPsudoSignature);
  process.exit(0);
})();

```

And you should get the base64 psbt with pseudo signature.

keiwn2 - August 31, 2024, 2:55 pm

So these scripts have worked for you before?

notagh0st - August 31, 2024, 3:09 pm

(Aug 31, 2024 02:55 PM)keiwn2 Wrote: [\[CITE\]](#) So these scripts have worked for you before? Yes, they work for me. I just tested them again before sending them to you.

keiwn2 - August 31, 2024, 3:15 pm

Ok I shall test them. But thank you for this.

I've created a script that generates a PSBT with a pseudo-signature, but I'm encountering issues with properly formatting the partialSig to ensure it's recognized in the scriptSig field by bitcoin-cli.

Could you help me adjust the script so that the signature appears correctly in the scriptSig field when the transaction is decoded?

I've created a script that generates a PSBT with a pseudo-signature, but I'm encountering issues with properly formatting the partialSig to ensure it's recognized in the scriptSig field by bitcoin-cli.

Could you help me adjust the script so that the signature appears correctly in the scriptSig field when the transaction is decoded?

I'm working on creating a Bitcoin transaction that includes a pseudo-signature using PSBT, but I'm facing challenges with getting it accepted by standard validation tools. The goal is to include a transaction in a block, even though the signature isn't cryptographically valid. Could you provide or help with a script or method to produce a correctly formatted PSBT that appears signed, so it can be manually included in a block by a mining pool? Your expertise would be greatly appreciated.