PSBT Howto for Bitcoin Core

Since Bitcoin Core 0.17, an RPC interface exists for Partially Signed Bitcoin Transactions (PSBTs, as specified in BIP 174).

This document describes the overall workflow for producing signed transactions through the use of PSBT, and the specific RPC commands used in typical scenarios.

PSBT in general

PSBT is an interchange format for Bitcoin transactions that are not fully signed yet, together with relevant metadata to help entities work towards signing it. It is intended to simplify workflows where multiple parties need to cooperate to produce a transaction. Examples include hardware wallets, multisig setups, and CoinJoin transactions.

Overall workflow

Overall, the construction of a fully signed Bitcoin transaction goes through the following steps:

- A **Creator** proposes a particular transaction to be created. They construct a PSBT that contains certain inputs and outputs, but no additional metadata.
- For each input, an **Updater** adds information about the UTXOs being spent by the transaction to the PSBT. They also add information about the scripts and public keys involved in each of the inputs (and possibly outputs) of the PSBT.
- **Signers** inspect the transaction and its metadata to decide whether they agree with the transaction. They can use amount information from the UTXOs to assess the values and fees involved. If they agree, they produce a partial signature for the inputs for which they have relevant key(s).
- A **Finalizer** is run for each input to convert the partial signatures and possibly script information into a final `scriptSig` and/or `scriptWitness`.
- An **Extractor** produces a valid Bitcoin transaction (in network format) from a PSBT for which all inputs are finalized.

Generally, each of the above (excluding Creator and Extractor) will simply add more and more data to a particular PSBT, until all inputs are fully signed. In a naive workflow, they all have to operate sequentially, passing the PSBT from one to the next, until the Extractor can convert it to a real transaction. In order to permit parallel operation, **Combiners** can be employed which merge metadata from different PSBTs for the same unsigned transaction.

The names above in bold are the names of the roles defined in BIP174. They're useful in understanding the underlying steps, but in practice, software and hardware implementations will typically implement multiple roles simultaneously.

PSBT in Bitcoin Core

RPCs

- `converttopsbt` **(Creator)** is a utility RPC that converts an unsigned raw transaction to PSBT format. It ignores existing signatures.
- `createpsbt` **(Creator)** is a utility RPC that takes a list of inputs and outputs and converts them to a PSBT with no additional information. It is equivalent to calling `createrawtransaction` followed by `converttopsbt`.
- `walletcreatefundedpsbt` **(Creator, Updater)** is a wallet RPC that creates a PSBT with the specified inputs and outputs, adds additional inputs and change to it to balance it out, and adds relevant metadata. In particular, for inputs that the wallet knows about (counting towards its normal or watch-only balance), UTXO information will be added. For outputs and inputs with UTXO information present, key and script information will be added which the wallet knows about. It is equivalent to running `createrawtransaction`, followed by `fundrawtransaction`, and `converttopsbt`.
- `walletprocesspsbt` **(Updater, Signer, Finalizer)** is a wallet RPC that takes as input a PSBT, adds UTXO, key, and script data to inputs and outputs that miss it, and optionally signs inputs. Where possible it also finalizes the partial signatures.
- `utxoupdatepsbt` **(Updater)** is a node RPC that takes a PSBT and updates it to include information available from the UTXO set (works only for SegWit inputs).
- `finalizepsbt` **(Finalizer, Extractor)** is a utility RPC that finalizes any partial signatures, and if all inputs are finalized, converts the result to a fully signed transaction which can be broadcast with `sendrawtransaction`.
- `combinepsbt` **(Combiner)** is a utility RPC that implements a Combiner. It can be used at any point in the workflow to merge information added to different versions of the same PSBT. In particular it is useful to combine the output of multiple Updaters or Signers.
- `joinpsbts` (Creator) is a utility RPC that joins multiple PSBTs together, concatenating the inputs and outputs. This can be used to construct CoinJoin transactions.
- `decodepsbt` is a diagnostic utility RPC which will show all information in a PSBT in human-readable form, as well as compute its eventual fee if known.
- `analyzepsbt` is a utility RPC that examines a PSBT and reports the current status of its inputs, the next step in the workflow if known, and if possible, computes the fee of the resulting transaction and estimates the final weight and feerate.

Workflows

Multisig with multiple Bitcoin Core instances For a quick start see Basic M-of-N multisig example using descriptor wallets and PSBTs. If you are using

legacy wallets feel free to continue with the example provided here.

Alice, Bob, and Carol want to create a 2-of-3 multisig address. They're all using Bitcoin Core. We assume their wallets only contain the multisig funds. In case they also have a personal wallet, this can be accomplished through the multiwallet feature - possibly resulting in a need to add `-rpcwallet=name` to the command line in case `bitcoin-cli` is used.

Setup: - All three call `getnewaddress` to create a new address; call these addresses *Aalice*, *Abob*, and *Acarol*. - All three call `getaddressinfo "X"`, with *X* their respective address, and remember the corresponding public keys. Call these public keys *Kalice*, *Kbob*, and *Kcarol*. - All three now run `addmultisigaddress 2 ["Kalice","Kbob","Kcarol"]` to teach their wallet about the multisig script. Call the address produced by this command *Amulti*. They may be required to explicitly specify the same addresstype option each, to avoid constructing different versions due to differences in configuration. - They also run `importaddress "Amulti" "" false` to make their wallets treat payments to *Amulti* as contributing to the watch-only balance. - Others can verify the produced address by running `createmultisig 2 ["Kalice","Kbob","Kcarol"]`, and expecting *Amulti* as output. Again, it may be necessary to explicitly specify the addresstype in order to get a result that matches. This command won't enable them to initiate transactions later, however. - They can now give out *Amulti* as address others can pay to.

Later, when *V* BTC has been received on *Amulti*, and Bob and Carol want to move the coins in their entirety to address *Asend*, with no change. Alice does not need to be involved. - One of them - let's assume Carol here - initiates the creation. She runs `walletcreatefundedpsbt [] {"Asend":V} 0 {"subtractFeeFromOutputs":[0], "includeWatching":true}`. We call the resulting PSBT *P*. *P* does not contain any signatures. - Carol needs to sign the transaction herself. In order to do so, she runs `walletprocesspsbt "P"`, and gives the resulting PSBT *P2* to Bob. - Bob inspects the PSBT using `decodepsbt "P2"` to determine if the transaction has indeed just the expected input, and an output to *Asend*, and the fee is reasonable. If he agrees, he calls `walletprocesspsbt "P2"` to sign. The resulting PSBT *P3* contains both Carol's and Bob's signature. - Now anyone can call `finalizepsbt "P3"` to extract a fully signed transaction *T*. - Finally anyone can broadcast the transaction using `sendrawtransaction "T"`.

In case there are more signers, it may be advantageous to let them all sign in parallel, rather than passing the PSBT from one signer to the next one. In the above example this would translate to Carol handing a copy of *P* to each signer separately. They can then all invoke `walletprocesspsbt "P"`, and end up with their individually-signed PSBT structures. They then all send those back to Carol (or anyone) who can combine them using `combinepsbt`. The last two steps (`finalizepsbt` and `sendrawtransaction`) remain unchanged.