```
BIP: 127
Layer: Applications
Title: Simple Proof-of-Reserves Transactions
Author: Steven Roose <steven@stevenroose.org>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0127
Status: Draft
Type: Standards Track
Created: 2019-01-28
License: CC0-1.0
```

Abstract

This BIP describes a simple way to construct proof-of-reserves transactions. This proposal formalizes a standard format for constructing such proofs, easing their construction with existing wallet infrastructure and enabling general proof-verification software. It relies on existing standards such as regular Bitcoin transaction serialization/validation and the BIP 174 PSBT format. The proposal also includes the description of a PSBT extension for a better user experience.

Copyright

Motivation

From the very early days in the history of Bitcoin, there have been companies managing bitcoins for their users. These users give up control over their coins in return for a certain service. Inevitably, there have been many cases of companies losing their users' bitcoins without timely disclosing such events to the public. Proofs of Reserves are a way for companies managing large amounts of bitcoins to prove ownership over a given amount of funds. The regular proof of control helps to ensure that no significant loss has occurred.

While the term proof-of-reserves is not new by any means, the procedure is not very common among high-value custodian companies. One of the reasons for this is that every company that wants to perform a proof-of-reserves has to construct its own way to do so. Accordingly, their users have to understand the construction of the proof in order to be able to verify it. This raises the bar of entry both for custodians and for users.

What this BIP is not doing

The proof-of-reserve construction described in this document has some known shortcomings, mostly with regards to its privacy properties. While there exists research about improved proof-of-reserves mechanisms that have much better privacy properties[1], this BIP intentionally only formalizes the de-facto existing method.

Specification

Our specification consists of two parts:

1. the format for the actual proofs
2. a file format used to package a set of proofs and relevant metadata

The final construction should have the following properties:

- flexible proof construction to support complex wallet infrastructures
- easy integration with existing wallet solutions (both hardware and software wallets)
- support for verification via a standard procedure, regardless of publisher of the proof
- proof prevents reuse of proofs by other parties by committing to a message
- allow validating that the issuer had the funds under his control at a certain block, regardless of what happened after that block

Proof Format

To allow for maximal compatibility with existing systems, proofs are formatted as regular Bitcoin transactions. However, one small adaptation to the transaction is made that has two functions:

1. make the transaction unspendable to avoid putting funds at risk
2. link the proof to the issuer of the proof to prevent copying proofs from other custodians

The resulting construction is a Bitcoin transaction with the following characteristics:

- The first input (the "commitment input")
  - MUST have the txid part of the previous outpoint set to the SHA-256 hash of the commitment message prefixed with "Proof-of-Reserves: "[2] and index 0.
- The remaining inputs
  - MUST have signatures that commit to the commitment input (e.g. using `SIGHASH_ALL`).
- The transaction MUST have a single output that is the exact sum of all the inputs, assuming the commitment input to have 0 value; this means the transaction has no miner fee.

[1]Dagher, Gaby G., Benedikt Bünz, Joseph Bonneau, Jeremy Clark, and Dan Boneh. "Provisions: Privacy-preserving proofs of solvency for Bitcoin exchanges." (2015).

[2]If the message is "Some Message", the txid part should be `SHA-256("Proof-of-Reserves: Some Message")` with the string encoded as UTF-8.

The existence of the first input (which is just a commitment hash) ensures that this transaction is invalid and can never be confirmed.

Proof File Format

In theory, the first part of the specification would be sufficient as a minimum viable standard. However, there are a number of motivations to extend the standard with an extra layer of metadata:

1. constructing and combining multiple proofs
   Having thousands of UTXOs spread across different offline and online wallets could make it difficult to construct a single proof transaction with all UTXOs. Allowing multiple proof transactions with the same commitment message and block number gives extra flexibility to custodians with complex wallet infrastructure without making the combined proof less secure.
2. metadata for verification
   Not all systems that will be used for verification have access to a full index of all transactions. However, proofs should be easily verifiable even after some of the UTXOs used in the proof are no longer unspent. Metadata present in the proof allows for relatively efficient verification of proofs even if no transaction index is available.
3. potential future improvements
   The extensible metadata format allows for amending the standard in the future. One potential improvement would be having UTXO set commitments. These would allow the proofs-of-reserves to come with accompanying proofs-of-inclusion of all used UTXOs in the UTXO set at the block of proof construction (making validation even more efficient).

The proposed proof-file format provides a standard way of combining multiple proofs and associated metadata. The specification of the format is in the Protocol Buffers[3] format.

```
syntax = "proto3";
import "google/protobuf/any.proto";

message OutputMeta {
    // Identify the outpoint.
    bytes txid = 1;
    uint32 vout = 2;

    // The block hash of the block where this output was created.
    bytes block_hash = 3;
}
```

[3]https://github.com/protocolbuffers/protobuf/

```
message FinalProof {
    // The proof transaction.  Should be able to be parsed like a regular
    // Bitcoin transaction.
    bytes proof_tx = 1;

    // The metadata of the ouputs used in the proof transaction.
    repeated OutputMeta output_metadata = 2;
}

message ProofOfReserves {
    // A version number for this format to enable extending it with
    // additional fields.
    uint32 version = 1;

    // The network magic for the network in which the proofs are valid.
    // 0xD9B4BEF9 for mainnet, 0x0709110B for testnet
    //TODO consider BIP44 coin type ids instead:
    // https://github.com/satoshilabs/slips/blob/master/slip-0044.md
    uint32 network_magic = 2;

    // The commitment message for this proof-of-reserves.
    // This message is global for all the proofs.
    string message = 3;

    // The block at which this proof is supposed to be validated.
    // Verification should take into account unspentness of outputs at this
    // block height.
    bytes block_hash = 4;

    // The set of final proof transactions with their output metadata.
    repeated FinalProof final_proofs = 5;

    // Reserved field that can potentially be used by proof-construction tools.
    // It can be ignored for verification.
    repeated google.protobuf.Any pending_proofs = 6;
}
```

The last field, `pending_proofs`, leaves open some space in the same file that can be used by proof-construction tools. This allows them to construct different proofs incrementally without having to switch between file formats.

PSBT (BIP 174) extension

The "commitment input" detailed in the proof format section does not spend an existing UTXO and thus shouldn't be signed (empty `scriptSig` and witness). This can cause some problems when signing this type of transactions. For example, hardware wallets often require the signer to provide information about all inputs of transactions they are signing, such as the previous output or previous transaction; this data obviously doesn't exist for the commitment inputs.

For most existing devices, it's possible to circumvent these requirements by providing dummy data or by instructing the device to ignore this specific input. However, there is still a UX problem. Because the hardware wallet device doesn't recognize the transaction as a proof-of-reserves transaction it will think it is signing a regular transaction that is spending all the money in the UTXOs. Most devices will ask for confirmation with a message along the lines of "Are you sure you want to send XXX BTC to address [...]?". This is not the best user experience.

An addition to the BIP 174 PSBT format could help signing devices to recognize proof-of-reserve transactions. The following field is added to the BIP 174 `INPUT` map:

- Type: Proof-of-reserves commitment `PSBT_IN_POR_COMMITMENT = 0x09`
  - Key: None. The key must only contain the 1 byte type.
    - `{0x09}`
  - Value: The UTF-8 encoded commitment message string for the proof-of-reserves.
    - `{porCommitment}`

Wallets processing an input that has this field set

- MUST make sure the txid of the previous outpoint is set to the SHA-256 hash of the prefixed commitment message string, as detailed above;
- MUST assume the input value to be 0 (without requiring the previous output or transaction to be provided);
- SHOULD display the commitment message to ask the user for confirmation before signing any inputs;
- SHOULD only provide signatures with a signature hash that commits to this input;
- SHOULD accept an empty `scriptSig` for this input (as if the `scriptPubKey` was `OP_TRUE`).

Compatibility

The proof transaction specification is based on the Bitcoin transaction serialization protocol and will thus always be compatible with serializers that can interpret Bitcoin transactions. The protobuf file format is custom to this BIP and has a version byte to enable updates while attempting to remain backwards compatible.

Implementations

A proof-of-concept implementation of the PSBT extension in the rust-bitcoin project can be found in the `psbt-por` branch here: https://github.com/stevenroose/rust-bitcoin/tree/psbt-por

A work-in-progress implementation of a tool that produces and verifies proofs in the described format can be found here: https://github.com/stevenroose/reserves

An implementation of the custom proof PSBTs is part of the BDK, and can be found here: https://crates.io/crates/bdk-reserves

Footnotes