```
BIP: 120
Layer: Applications
Title: Proof of Payment
Author: Kalle Rosenbaum <kalle@rosenbaum.se>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0120
Status: Withdrawn
Type: Standards Track
Created: 2015-07-28
```

Abstract

This BIP describes a system called Proof of Payment, PoP. It is used to prove that a wallet has the credentials that were used to sign a previously generated transaction. Or simply put, it lets you prove that you have made a payment.

Motivation

There are several scenarios in which it would be useful to prove that you have paid for something. For example:

- A pre-paid hotel room where your PoP functions as a key to the door.
- An online video rental service where you pay for a video and watch it on any device.
- An ad-sign where you pay in advance for e.g. 2 weeks exclusivity. During this period you can upload new content to the sign whenever you like using PoP.
- Log in to a pay site using a PoP.
- A parking lot you pay for monthly and the car authenticates itself using PoP.
- A lottery where all participants pay to the same address, and the winner is selected among the transactions to that address. You exchange the prize for a PoP for the winning transaction.

With Proof of Payment, these use cases can be achieved without any personal information (user name, password, e-mail address, etc) being involved.

Rationale

Desirable properties:

1. A PoP should be generated on demand.
2. It should only be usable once to avoid issues due to theft.
3. It should be able to create a PoP for any payment, regardless of script type (P2SH, P2PKH, etc.).
4. It should prove that you have enough credentials to unlock all the inputs of the proven transaction.

5. It should be easy to implement by wallets and servers to ease adoption.

Current methods of proving a payment:

- In BIP0070, the PaymentRequest together with the transactions fulfilling the request makes some sort of proof. However, it does not meet 1, 2 or 4 and it obviously only meets 3 if the payment is made through BIP0070. Also, there's no standard way to request/provide the proof. If standardized it would probably meet 5.
- Signing messages, chosen by the server, with the private keys used to sign the transaction. This could meet 1 and 2 but probably not 3. This is not standardized either. 4 Could be met if designed so.

If an input script type is P2SH, any satisfying script should do, just as if it was a payment. For M-of-N multisig scripts, that would mean that any set of M keys should be sufficient, not necessarily the same set of M keys that signed the transaction. This is important because strictly demanding the same set of M keys would defeat the purpose of a multisig address.

Specification

Data structure

A proof of payment for a transaction T, here called PoP(T), is used to prove that one has ownership of the credentials needed to unlock all the inputs of T. It has the exact same structure as a bitcoin transaction with the same inputs as T and in the same order as in T, but with each sequence number set to 0. There is exactly one output, here called the pop output, with value 0. The pop output must have the following format:

`OP_RETURN`

Field	Size [B]	Description
<version>	2	Version, little endian, currently 0x01 0x00
<txid>	32	The transaction to prove
<nonce>	6	Random data

The lock_time of the PoP must be set to 499999999 to prevent the PoP from being included in a block, should it appear on the bitcoin p2p network. This is also the reason for setting the sequence numbers to 0, since sequence number of ffffffff would make lock_time ineffective. This specification demands that all input sequence numbers are 0, not just one of them, which would be sufficient to make lock_time effective. This is for simplicity reasons.

An illustration of the PoP data structure and its original payment is shown below.

```
 T
+----------------------------------------------+
|inputs                | outputs               |
|       Value,Sequence | Value,Script          |
+----------------------------------------------+
|input0 1,ffffffff     | 0,pay to A            |
|input1 3,ffffffff     | 2,OP_RETURN <some data> |
|input2 4,ffffffff     | 1,pay to B            |
|                      | 4,pay to C            |
+----------------------------------------------+

 PoP(T)
+----------------------------------------------------------+
| inputs               | outputs                           |
|       Value,Sequence | Value,Script                      |
+----------------------------------------------------------+
|input0 1,00000000     | 0,OP_RETURN <version> <txid> <nonce> |
|input1 3,00000000     |                                   |
|input2 4,00000000     |                                   |
+----------------------------------------------------------+
| lock_time=499999999                                      |
+----------------------------------------------------------+
```

The PoP is signed using the same signing process that is used for bitcoin transactions.

The purpose of the nonce is to make it harder to use a stolen PoP; Once the PoP has reached the server, that PoP is useless since the server will generate a new nonce for every PoP request.

Process

1. A proof of payment request is sent from the server to the wallet. The PoP request contains:
   (a) a random nonce
   (b) a destination where to send the PoP, for example a https URL
   (c) data hinting the wallet which transaction to create a proof for. For example:
      - txid, if known by the server
      - PaymentRequest.PaymentDetails.merchant_data (in case of a BIP0070 payment)
      - amount, label, message or other information from a BIP0021 URI
2. The wallet identifies a transaction T, if possible. Otherwise it asks the user to select among the ones that match the hints in 1.iii.
3. The wallet creates an unsigned PoP (UPoP) for T, and asks the user to sign it.
4. The user confirms

5. The UPoP(T) is signed by the wallet, creating PoP(T).
6. The PoP is sent to the destination in 1.ii.
7. The server receiving the PoP validates it and responds with “valid” or “invalid”.
8. The wallet displays the response in some way to the user.

Remarks:

- The method of transferring the PoP request at step 1 is not specified here. Instead that is specified in separate specifications, see BIP0121.
- The nonce must be randomly generated by the server for every new PoP request.

Validating a PoP

The server needs to validate the PoP and reply with "valid" or "invalid". That process is outlined below. If any step fails, the validation is aborted and "invalid" is returned:

1. Check the format of the PoP. It must pass normal transaction checks, except that the inputs may already be spent.
2. Check that lock_time is 499999999.
3. Check that there is exactly one output. This output must have value 0 and conform to the OP_RETURN output format outlined above.
4. Check that the nonce is the same as the one requested.
5. Check that the inputs of the PoP are exactly the same as in transaction T, except that the sequence numbers must all be 0. The ordering of the inputs must also be the same as in T.
6. Run the scripts of all the inputs. All scripts must return true.
7. Check that the txid in the PoP output is the transaction you actually want proof for. If you don’t know exactly what transaction you want proof for, check that the transaction actually pays for the product/service you deliver.
8. Return "valid".

Security considerations

- Someone can intercept the PoP-request and change any parameter in it. These can be mitigated by using secure connections. Examples of tampered parameters:
  - Pop destination - Stealing your PoP.
  - label - Trick you to sign an unintended pop or set a label that your wallet doesn't have any record for, resulting in a broken service. Always check the PoP before signing.
  - nonce - Your pop will not validate on server.
- Someone can steal a PoP, for example by tampering with the PoP request, and try to use the service hoping to get a matching nonce. Probability per try: 1/(2^48). The server should have a mechanism for detecting a brute

force attack of this kind, or at least slow down the process by delaying the PoP request by some 100 ms or so.

- Even if a wallet has no funds it might still be valuable as a generator for PoPs. This makes it important to keep the security of the wallet after it has been emptied.
- Transaction malleability may cause the server to have a different transaction id for a payment than the client's wallet. In that case the wallet will not be able to prove the transaction to the server. Wallets should not rely on the transaction id of the outgoing transaction. Instead it should listen for the transaction on the network and put that in its list of transactions.

Reference implementation

PoP Demo server on GitHub

PoP-enabled Mycelium fork on GitHub

References

BIP0021: URI Scheme

BIP0070: Payment Protocol

BIP0121: Proof of Payment URI scheme