```
BIP: 121
Layer: Applications
Title: Proof of Payment URI scheme
Author: Kalle Rosenbaum <kalle@rosenbaum.se>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0121
Status: Withdrawn
Type: Standards Track
Created: 2015-07-27
```

Abstract

This is a proposal for a URI scheme to be used in the Proof of Payment process.

Motivation

To make a Proof of Payment, the party that wants the proof needs to transfer a Proof of Payment request to the wallet software of the other party. To facilitate that transfer, a new URI scheme representing the PoP request is proposed. This URI can then be encoded in QR images or be sent over NFC in order to transfer it to the wallet.

Specification

The specification is the same as BIP0021, with the following differences:

- The URI scheme is `btcpop` instead of `bitcoin`
- The path component, i.e. the address part, is always empty.
- A mandatory `p` parameter whose value contains the destination for the PoP. This could for example be a `https:` URL or a `mailto:` URI.
- A mandatory `n` parameter representing the nonce, base58 encoded.
- An optional `txid` parameter containing the Base58 encoded hash of the transaction to prove.

Just as in BIP0021, elements of the query component may contain characters outside the valid range. These must first be encoded according to UTF-8, and then each octet of the corresponding UTF-8 sequence must be percent-encoded as described in RFC 3986.

All parameters except `p` and `n` are hints to the wallet on which transaction to create a PoP for.

The extensibility of BIP0021 applies to this scheme as well. For example, a `date` parameter or a `toaddr` parameter might be useful. `req-*` parameters are also allowed and obey the same rules as in BIP0021, clients not supporting a `req-*` parameter must consider the URI invalid.

Keep URIs short

Implementations should keep the URIs as short as possible. This is because it makes QR decoding more stable. A camera with a scratched lens or low resolution may run into problems scanning huge QR codes. This is why the `txid` parameter is encoded in Base58 instead of the classic hex encoded string. We get away with 44 characters instead of 64. Also, the `nonce` parameter is Base58 encoded for the same reason.

Interpretation

Transaction hints

The wallet processing the URI must use the hints in the PoP request to filter its transaction set. The `label`, `amount` and `message` parameters must, if present in the URI, exactly match the data associated with the original payment according to the following table:

`btcpop:` URI parameter	`bitcoin:` URI parameter	BIP70 PaymentDetails data
`label`	`label`	`memo`
`amount`	`amount`	`sum of outputs.amount`
`message`	`message`	-

The `txid` parameter value must match the transaction hash of the payment.

After filtering, the resulting transaction set is displayed to the user who selects one of them to prove. An implementation could also automatically select a transaction in the filtered set, but there must still be a way for the user to select freely among the matching transactions. If the filtered set is empty, no transaction fits the hints and a message about that is presented to the user. If the filtered set contains exactly one transaction, which is preferable, that transaction can be automatically selected.

As a fallback, there must also be a way for the user to select any transaction from the wallet regardless of the transaction hints. This can be useful if the metadata of the wallet is lost, possibly due to a restore from backup.

PoP destination `p`

The `p` parameter value is the destination where to send the PoP to. This destination is typically a `https:` URL or a `http:` URL, but it could be any type of URI, for example `mailto:`. To keep `btcpop:` URIs short, users should not make their `p` parameter unnecessarily long.

`http:` and `https:` URLs Wallet implementations must support the `http:` and `https:` schemes in which case `POST` method must be used. The PoP is sent

as a binary serialized transaction. The content type of the POST request must be set to `application/bitcoin-pop`

Examples

Send PoP for a transaction with label "video 42923" to https://www.example.com/pop/352, using nonce `0x73 0xd5 0x1a 0xbb 0xd8 0x9c`:

```
btcpop:?p=https://www.example.com/pop/352&n=zgWTm8yH&label=video%2042923
```

Send PoP through mail using mailto:pop@example.com?subject=pop444, amount is 13370000 satoshis, nonce is `0x6f 0xe 0xfb 0x68 0x92 0xf9`. Note that the `?` before `subject` is OK according to RFC3986, since the query part starts from the first `?`:

```
btcpop:?p=mailto:pop@example.com?subject%3Dpop444&n=xJdKmEbr&amount=0.1337
```

Send PoP for transaction with id `cca7507897abc89628f450e8b1e0c6fca4ec3f7b34cccf55f3f531c659ff4d79` to pizza place at http://pizza.example.com/pop/laszlo111 using nonce `0xfc 0xcc 0x2c 0x35 0xf0 0xb8`

```
btcpop:?p=http://pizza.example.com/pop/laszlo111&n=3AtNpVrPh&txid=Emt9MPvt1joznqHy5eEHkNtcu
```

Reference implementation

PoP Demo server on GitHub

PoP-enabled Mycelium fork on GitHub

References

BIP0021: URI Scheme

BIP0120: Proof of Payment

RFC3986: Uniform Resource Identifier (URI): Generic Syntax