```
BIP: 175
Layer: Applications
Title: Pay to Contract Protocol
Author: Omar Shibli <omar@commerceblock.com>
        Nicholas Gregory <nicholas@commerceblock.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0175
Status: Rejected
Type: Informational
Created: 2017-07-17
License: BSD-2-Clause
```

Abstract

Utilizing hierarchical deterministic wallets as described in BIP-0032 and the "Purpose Field" in BIP-0043, this document specifies the multiparty pay-to-contract key derivation scheme outlined by Ilja Gerhardt and Timo Hanke.[0]

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Motivation

A Bitcoin transaction represents a "real world" contract between two parties transferring value. Counterparties in a business interaction traditionally keep track of a payment with bills (invoices) and receipts. Delivery of a good is made by the payee once the payer has signed the receipt, agreeing to pay for the items on the invoice. Gerhardt and Hanke [0] formulate this interaction within the confines of the Bitcoin protocol using homomorphic payment addresses and the multiparty pay-to-contract protocol.

The protocol is constructed in such a way that all parties have cryptographic proof of both who is being paid and for what. Using the technique described in this BIP, an address can be provably derived from the terms of a contract and the payee's public key. This derivation scheme does not bloat the UTXO and is completely hidden to network participants; the derived address looks like any other P2(W)PKH or P2(W)SH address. Redemption of the funds requires knowledge of the contract and the payee's private key.

This scheme utilizes the foundations of BIP-0032, providing a consistent way for preexisting wallet developers to implement the specification.

Specification

This key derivation scheme requires two parties: a payer (customer) and a payee (merchant). The customer submits to the merchant a purchase request, specifying what goods/services they would like to buy. From the purchase request the merchant constructs an invoice (contract), specifying the billable items and total amount to be paid. The merchant must give this contract alongside a “payment base” extended public key to the customer. Given this information, the customer will be able to fulfill the contract by generating the public key of the payment address associated with the contract and the payment base, then sending the funds there.

We define the following levels in BIP32 path:

```
m / purpose' / coin_type' / contract_hash
```

`contract_hash` consists of multiple levels.

Apostrophe in the path indicates that BIP32 hardened derivation is used.

We define the following extended public keys:

Payment base denoted as `payment_base`:

```
   m / purpose' / coin_type'
```

Payment address denoted as `payment_address`:

```
   m / purpose' / coin_type' / contract_hash
or
m / payment_base / contract_hash
```

Each level has special meaning described in the chapters below.

Purpose

Purpose is a constant set to `175'` (or `0x800000AF`) following the BIP-0043 recommendation. It indicates that the subtree of this node is used according to this specification.

```
m / 175' / *
```

Hardened derivation is used at this level.

Coin type

The coin type field is identical to the same field in BIP-0044.

Hardened derivation is used at this level.

Payment address generation

For a given contract documents denoted by c_1,...,c_n, payment base extended public key denoted by `payment_base`, and cryptographic hash function denoted by `h`.

1. Compute cryptographic hashes for all contract documents, by applying the hash function.

```
h(c1),...,h(cn)
```

2. Sort all hashes lexicographically.

```
hash_1,...,hash_n
```

3. Prepend payment_base and concatenate the sorted hashes and apply the hash function.

```
h(payment_base+hash_1+...+hash_n)
```

4. Compute a partial BIP32 derivation path from the combined hash as defined in Hash to Partial Derivation Path Mapping procedure below.

```
contract_hash
```

5. Prepend `payment_base` to contract_hash derivation path.

```
payment_base / contract_hash
```

6. Compute public extended key from the derivation path in step 5.

7. Compute address of the public extended key (P2PKH) from step 6.

Payment address verification

For a given Bitcoin address, `payment_base` extended public key, contract documents denoted by c_1,...,c_n, and cryptographic hash function denoted by `h`, we can verify the integrity of the address by the following steps:

1. Compute contract address from the given inputs as described in Contract Address Generation section.

2. Compare the computed address from step 1 with the given Bitcoin address as an input.

Redemption

The merchant is able to construct the private key offline using the method described in the Payment Address Generation section. The merchant should actively monitor the blockchain for the payment to the payment address. Because the address is generated from the payment base and the contract, the merchant must implicitly agree to those terms in order to spend the funds. The act of making the payment to that address thus serves as a receipt for the customer.

Hash to partial derivation path mapping

At this section, we define hash to partial BIP32 derivation path mapping procedure that maps between an arbitrary hex number to a partial BIP32 derivation path.

For a given hex number, do the following:

1. Partition hex number into parts, each part length is 4 chars.
2. Convert each part to integer in decimal format.
3. Concatenate all numbers with slash `/`.

Examples

For the following given inputs:

```
 master private extended key:
xprv9s21ZrQH143K2JF8RafpqtKiTbsbaxEeUaMnNHsm5o6wCW3z8ySyH4UxFVSfZ8n7ESu7fgir8imbZKLYVBxFPND1
coin type:
0
```

we can compute payment base as follows:

```
 payment base derivation path:
m/175'/0'
contract base public extended key:
xpub6B3JSEWjqm5GgfzcjPwBixxLPzi15pFM3jq4E4yCzXXUFS5MFdXiSdw7b5dbdPGHuc7c1V4zXbbFRtc9G1njMUt9
```

In the below examples, we are going to use SHA256 as a cryptographic hash function, and the above contract base public key.

Payment address generation As an input, we have a contract that consists of two documents, below are contents:

document 1:

```
 bar
```

document 2:

```
 foo
```

1. Apply the hash function:

```
 document 1:
fcde2b2edba56bf408601fb721fe9b5c338d10ee429ea04fae5511b68fbf8fb9
document 2:
2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae
```

2. Sort all hashes lexicographically:

```
 2c26b46b68ffc68ff99b453c1d30413413422d706483bfa0f98a5e886266e7ae
fcde2b2edba56bf408601fb721fe9b5c338d10ee429ea04fae5511b68fbf8fb9
```

3. Concatenate hashes and apply the hash function.

```
 concatenated hash: payment_base
xpub6B3JSEWjqm5GgfzcjPwBixxLPzi15pFM3jq4E4yCzXXUFS5MFdXiSdw7b5dbdPGHuc7c1V4zXbbFRtc9G1njMUt9
combined hash:
310057788c6073640dc222466d003411cd5c1cc0bf2803fc6ebbfae03ceb4451
```

4. Compute the partial BIP32 derivation path of the combined hash.

```
12544/22392/35936/29540/3522/8774/27904/13329/52572/7360/48936/1020/28347/64224/15595/1748
```

5. Prepend `payment_base` to `contract_hash` derivation path.

```
contract_base_pub/12544/22392/35936/29540/3522/8774/27904/13329/52572/7360/48936/1020/2834
or
m/175'/0'/12544/22392/35936/29540/3522/8774/27904/13329/52572/7360/48936/1020/28347/64224/1
```

6. Compute public extended key.

```
xpub6hefaATTG5LbcwyPDvmNfnkyzefoM2TJDoo5astH7Gvs1g8vZURviBWvAvBnWc2CNb8ybJ6mDpnQYVsvNSZ3oU
```

7. Compute address of the public extended key (P2PKH).

```
1C7f322izqMqLzZzfzkPAjxBzprxDi47Yf
```

Verification example (negative test) Similar to the input above, except this time we have a contract that consists of one document, below is the content:

document 1:

```
baz
```

1. Apply the hash function.

```
baa5a0964d3320fbc0c6a922140453c8513ea24ab8fd0577034804a967248096
```

2. Prepend payment_base

```
xpub6B3JSEWjqm5GgfzcjPwBixxLPzi15pFM3jq4E4yCzXXUFS5MFdXiSdw7b5dbdPGHuc7c1V4zXbbFRtc9G1njMU
```

2. Apply hash function

```
3a08605829413ce0bf551b08d21e4a28dbda6e407f90eff1c448e839050c73a1
```

3. Compute the partial derivation path.

```
5338/54412/19213/962/30664/62597/11873/59874/56779/24089/54550/19585/28087/36422/18666/175
```

4. Prepend contract_base$_{\text{pub}}$ to contract_hash derivation path.

```
contract_base_pub/5338/54412/19213/962/30664/62597/11873/59874/56779/24089/54550/19585/280
or
m/175'/0'/5338/54412/19213/962/30664/62597/11873/59874/56779/24089/54550/19585/28087/36422/
```

5. Compute public extended key.

```
xpub6h9k2KqsMpwghxt7naj1puhGV1ZDC88sxvpYN1HibCf8yQZdPsuhYmmvdK32Kf2Lb3rS1sV8UcZ1f84DJEiXuV
```

7. Compute address of the public extended key (P2PKH).

```
1QGe5LaDMAmHeibJbZBmZqhQDZSp7QCqSs
```

8. As expected the address doesn't match the Bitcoin address from the last example `1C7f322izqMqLzZzfzkPAjxBzprxDi47Yf`.

Verification operation will succeed only if we use identical documents to ones that have been used in the contract address generation.

Compatibility

This specification is not backward compatible with BIP32 specification, the proposed derivation scheme in this BIP is a BIP32 compliant. Communication between payer and payee as well as hashing the contract and generating the path requires significant modification to the wallet.

Reference implementations

- Reference wallet implementation, based on Copay project : https://github.com/commerceblock/copay (pull_request)
- Reference implementation to Hash to Partial Derivation Path Mapping in javascript (https://github.com/commerceblock/pay-to-contract-lib)

Reference

- BIP32 - Hierarchical Deterministic Wallets
- BIP43 - Purpose Field for Deterministic Wallets
- BIP44 - Multi-Account Hierarchy for Deterministic Wallets
- Homomorphic Payment Addresses and the Pay-to-Contract Protocol

Copyright