```
BIP: 69
Layer: Applications
Title: Lexicographical Indexing of Transaction Inputs and Outputs
Author: Kristov Atlas <kristov@openbitcoinprivacyproject.org>
Editor: Daniel Cousens <bips@dcousens.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0069
Status: Proposed
Type: Informational
Created: 2015-06-12
License: PD
```

Abstract

Currently there is no standard for bitcoin wallet clients when ordering transaction inputs and outputs. As a result, wallet clients often have a discernible blockchain fingerprint, and can leak private information about their users. By contrast, a standard for non-deterministic sorting could be difficult to audit. This document proposes deterministic lexicographical sorting, using hashes of previous transactions and output indices to sort transaction inputs, as well as values and scriptPubKeys to sort transaction outputs.

Copyright

Motivation

Currently, there is no clear standard for how wallet clients ought to order transaction inputs and outputs. Since wallet clients are left to their own devices to determine this ordering, they often leak information about their users' finances. For example, a wallet client might naively order inputs based on when addresses were added to a wallet by the user through importing or random generation. Many wallets will place spending outputs first and change outputs second, leaking information about both the sender and receiver's finances to passive blockchain observers. Such information should remain private not only for the benefit of consumers, but in higher order financial systems must be kept secret to prevent fraud. A researcher recently demonstrated this principle when he detected that Bitstamp leaked information when creating exchange transactions, enabling potential espionage among traders. [1]

One way to address these privacy weaknesses is by randomizing the order of inputs and outputs. [2] After all, the order of inputs and outputs does not impact the function of the transaction they belong to, making random sorting viable. Unfortunately, it can be difficult to prove that this sorting process is genuinely randomly sorted based on code or run-time analysis, especially if the software is closed source. A malicious software developer can abuse the ordering of inputs

and outputs as a side channel of leaking information. For example, if an attacker can patch a victim's HD wallet client to order inputs and outputs based on the bits of a master private key, then the attacker can eventually steal all of the victim's funds by monitoring the blockchain. Non-deterministic methods of sorting are difficult to audit because they are not repeatable.

The lack of standardization between wallet clients when ordering inputs and outputs can yield predictable quirks that characterize particular wallet clients or services. Such quirks create unique fingerprints that a privacy attacker can employ through simple passive blockchain observation.

The solution is to create an algorithm for sorting transaction inputs and outputs that is deterministic. Since it is deterministic, it should also be unambiguous — that is, given a particular transaction, the proper order of inputs and outputs should be obvious. To make this standard as widely applicable as possible, it should rely on information that is downloaded by both full nodes (with or without typical efficiency techniques such as pruning) and SPV nodes. In order to ensure that it does not leak confidential data, it must rely on information that is publicly accessible through the blockchain. The use of public blockchain information also allows a transaction to be sorted even when it is a multi-party transaction, such as in the example of a CoinJoin.

Specification

Applicability

This BIP applies to any transaction for which the order of its inputs and outputs does not impact the transaction's function. Currently, this refers to any transaction that employs the SIGHASH_ALL signature hash type, in which signatures commit to the exact order of inputs and outputs. Transactions that use SIGHASH_ANYONECANPAY and/or SIGHASH_NONE may include inputs and/or outputs that are not signed; however, compliant software should still emit transactions with lexicographically sorted inputs and outputs, even though they may later be modified by others.

In the event that future protocol upgrades introduce new signature hash types, compliant software should apply the lexicographical ordering principle analogously.

While out of scope of this BIP, protocols that do require a specified order of inputs/outputs (e.g. due to use of SIGHASH_SINGLE) should consider the goals of this BIP and how best to adapt them to the specific needs of those protocols.

Lexicographical Ordering

Lexicographical ordering is an algorithm for comparison used to sort two sets based on their cartesian order within their common superset. Lexicographic order is also often known as alphabetical order, or dictionary order.

Common implementations include:

- ‘std::lexicographical_compare‘ in C++ [5]
- ‘cmp‘ in Python 2.7
- ‘memcmp‘ in C [6]
- ‘Buffer.compare‘ in Node.js [7]

For more information, see the wikipedia entry on Lexicographical order. [8]

N.B. All comparisons do not need to operate in constant time since they are not processing secret information.

Transaction Inputs

Transaction inputs are defined by the hash of a previous transaction, the output index of a UTXO from that previous transaction, the size of an unlocking script, the unlocking script, and a sequence number. [3] For sorting inputs, the hash of the previous transaction and the output index within that transaction are sufficient for sorting purposes; each transaction hash has an extremely high probability of being unique in the blockchain — this is enforced for coinbase transactions by BIP30 — and output indices within a transaction are unique. For the sake of efficiency, transaction hashes should be compared first before output indices, since output indices from different transactions are often equivalent, while all bytes of the transaction hash are effectively random variables.

Previous transaction hashes (in reversed byte-order) are to be sorted in ascending order, lexicographically. In the event of two matching transaction hashes, the respective previous output indices will be compared by their integer value, in ascending order. If the previous output indices match, the inputs are considered equal.

Transaction malleability will not negatively impact the correctness of this process. Even if a wallet client follows this process using unconfirmed UTXOs as inputs and an attacker modifies the blockchain’s record of the hash of the previous transaction, the wallet client will include the invalidated previous transaction hash in its input data, and will still correctly sort with respect to that invalidated hash.

Transaction Outputs

A transaction output is defined by its scriptPubKey and amount. [3] For the sake of efficiency, amounts should be compared first for sorting, since they contain fewer bytes of information (8 bytes) compared to a standard P2PKH scriptPubKey (25 bytes). [4]

Transaction output amounts (as 64-bit unsigned integers) are to be sorted in ascending order. In the event of two matching output amounts, the respective output scriptPubKeys (as a byte-array) will be compared lexicographically, in ascending order. If the scriptPubKeys match, the outputs are considered equal.

Examples

Transaction 0a6a357e2f7796444e02638749d9611c008b253fb55f5dc88b739b230ed0c4c3:

Inputs:

```
0: 0e53ec5dfb2cb8a71fec32dc9a634a35b7e24799295ddd5278217822e0b31f57[0]
1: 26aa6e6d8b9e49bb0630aac301db6757c02e3619feb4ee0eea81eb1672947024[1]
2: 28e0fdd185542f2c6ea19030b0796051e7772b6026dd5ddccd7a2f93b73e6fc2[0]
3: 381de9b9ae1a94d9c17f6a08ef9d341a5ce29e2e60c36a52d333ff6203e58d5d[1]
4: 3b8b2f8efceb60ba78ca8bba206a137f14cb5ea4035e761ee204302d46b98de2[0]
5: 402b2c02411720bf409eff60d05adad684f135838962823f3614cc657dd7bc0a[1]
6: 54ffff182965ed0957dba1239c27164ace5a73c9b62a660c74b7b7f15ff61e7a[1]
7: 643e5f4e66373a57251fb173151e838ccd27d279aca882997e005016bb53d5aa[0]
8: 6c1d56f31b2de4bfc6aaea28396b333102b1f600da9c6d6149e96ca43f1102b1[1]
9: 7a1de137cbafb5c70405455c49c5104ca3057a1f1243e6563bb9245c9c88c191[0]
10: 7d037ceb2ee0dc03e82f17be7935d238b35d1deabf953a892a4507bfbeeb3ba4[1]
11: a5e899dddb28776ea9ddac0a502316d53a4a3fca607c72f66c470e0412e34086[0]
12: b4112b8f900a7ca0c8b0e7c4dfad35c6be5f6be46b3458974988e1cdb2fa61b8[0]
13: bafd65e3c7f3f9fdfdc1ddb026131b278c3be1af90a4a6ffa78c4658f9ec0c85[0]
14: de0411a1e97484a2804ff1dbde260ac19de841bebad1880c782941aca883b4e9[1]
15: f0a130a84912d03c1d284974f563c5949ac13f8342b8112edff52971599e6a45[0]
16: f320832a9d2e2452af63154bc687493484a0e7745ebd3aaf9ca19eb80834ad60[0]
```

Outputs:

```
0:     400057456     76a9144a5fba237213a062f6f57978f796390bdcf8d01588ac
1:     40000000000     76a9145be32612930b8323add2212a4ec03c1562084f8488ac
```

Transaction 28204cad1d7fc1d199e8ef4fa22f182de6258a3eaafe1bbe56ebdcacd3069a5f

Inputs:

```
0: 35288d269cee1941eaebb2ea85e32b42cdb2b04284a56d8b14dcc3f5c65d6055[0]
1: 35288d269cee1941eaebb2ea85e32b42cdb2b04284a56d8b14dcc3f5c65d6055[1]
```

Outputs:

```
0:     100000000     41046a0765b5865641ce08dd39690aade26dfbf5511430ca428a3089261361cef170e3929
1:     2400000000     41044a656f065871a353f216ca26cef8dde2f03e8c16202d2e8ad769f02032cb86a5eb5e
```

Discussion

- [Bitcoin-development] Lexicographical Indexing of Transaction Inputs and Outputs
- [Bitcoin-development] [RFC] Canonical input and output ordering in transactions

References

- 1: Bitstamp Info Leak

- 2: OBPP Random Indexing as Countermeasure
- 3: Mastering Bitcoin
- 4: Bitcoin Wiki on Script
- 5: std::lexicographical_compare
- 6: memcmp
- 7: Buffer.compare
- 8: Lexicographical order

Implementations

- Electrum
- BitcoinJS
- BitcoinJS Test Fixtures
- NodeJS
- Blockchain.info public beta
- Btcsuite

Acknowledgements

Danno Ferrin <danno@numisight.com>, Sergio Demian Lerner <sergiolerner@certimix.com>, Justus Ranvier <justus@openbitcoinprivacyproject.org>, and Peter Todd <pete@petertodd.org> contributed to the design and motivations for this BIP. A similar proposal was submitted to the Bitcoin-dev mailing list independently by Rusty Russell <rusty@rustcorp.com.au>