```
BIP: 134
Layer: Consensus (hard fork)
Title: Flexible Transactions
Author: Tom Zander <tomz@freedommail.ch>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0134
Status: Rejected
Type: Standards Track
Created: 2016-07-27
License: CC-BY-SA-4.0
         OPL
```

Abstract

This BIP describes the next step in making Bitcoin's most basic element, the transaction, more flexible and easier to extend. At the same time this fixes all known cases of malleability and resolves significant amounts of technical debt.

Summary

Flexible Transactions uses the fact that the first 4 bytes in a transaction determine the version and that the majority of the clients use a non-consensus rule (a policy) to not accept transaction version numbers other than those specifically defined by Bitcoin. This BIP chooses a new version number, 4, and defines that the data following the bytes for the version is in a format called Compact Message Format (CMF). CMF is a flexible, token based format where each token is a combination of a name, a format and a value. Because the name is added we can skip unused tokens and we can freely add new tokens in a simple manner in future. Soft fork upgrades will become much easier and cleaner this way.

This protocol upgrade cleans up past soft fork changes like BIP68 which reuse existing fields and do them in a better to maintain and easier to parse system. It creates the building blocks to allow new features to be added much cleaner in the future.

It also shows to be possible to remove signatures from transactions with minimal upgrades of software and still maintain a coherent transaction history. Tests show that this can reduce space usage to about 75%.

Motivation

After 8 years of using essentially the same transaction version and layout Bitcoin is in need of an upgrade and lessons learned in that time are taking into account when designing it. The most important detail is that we have seen a need for more flexibility. For instance when the 'sequence' fields were introduced in the old transaction format, and later deprecated again, the end result was that all

transactions still were forced to keep those fields and grow the blockchain while they all were set to the default value.

The way towards that flexibility is to use a generic concept made popular various decades ago with the XML format. The idea is that we give each field a name and this means that new fields can be added or optional fields can be omitted from individual transactions. Some other ideas are the standardization of data-formats (like integer and string encoding) so we create a more consistent system. One thing we shall not inherit from XML is its text-based format. Instead we use the Compact Message Format (CMF) which is optimized to keep the size small and fast to parse.

Token based file-formats are not new, systems like XML and HTMl use a similar system to allow future growth and they have been quite successful for decades in part because of this property.

Next to that this protocol upgrade will re-order the data-fields which allows us to cleanly fix the malleability issue which means that future technologies like Lightning Network will depend on this BIP being deployed.

At the same time, due to this re-ordering of data fields, it becomes very easy to remove signatures from a transaction without breaking its tx-id, which is great for future pruning features.

Features

- Fixes malleability
- Linear scaling of signature checking
- Very flexible future extensibility
- Makes transactions smaller
- Supports the Lightning Network

Additionally, in the v4 (flextrans) format we add the support for the following proofs;

- input amount. Including the amount means we sign this transaction only if the amount we are spending is the one provided. Wallets that do not have the full UTXO DB can safely sign knowing that if they were lied to about the amount being spent, their signature is useless.
- scriptBase is the combined script of input and output, without signatures naturally. Providing this to a hardware wallet means it knows what output it is spending and can respond properly. Including it in the hash means its signature would be broken if we lied..
- Double spent-proof. Should a node detect a double spent he can notify his peers about this fact. Instead of sending the entire transactions, instead he sends only a proof. The node needs to send two pairs of info that proves that in both transactions the CTxIn are identical.

Tokens

In the compact message format we define tokens and in this specification we define how these tokens are named, where they can be placed and which are optional. To refer to XML, this specification would be the schema of a transaction.

CMF tokens are triplets of name, format (like PositiveInteger) and value. Names in this scope are defined much like an enumeration where the actual integer value (id, below) is equally important to the written name. If any token found that is not covered in the next table it will make the transaction that contains it invalid.

Name	id	Format	Default Value	Description
TxEnd	0	BoolTrue	Required	A marker that is the end of the transaction
TxInPrevHash	1	ByteArray	Required	TxId we are spending
TxPrevIndex	2	Integer	0	Index in prev tx we are spending (applied t
TxInputStackItem	3	ByteArray		A 'push' of the input script
TxInputStackItemContinued	4	ByteArray	&nsbp;	Another section for the same input
TxOutValue	5	Integer	Required	Amount of Satoshis to transfer
TxOutScript	6	ByteArray	Required	The output script
TxRelativeBlockLock	7	Integer	Optional	Part of the input stating the amount of blo
TxRelativeTimeLock	8	Integer	Optional	Part of the input stating the amount of tim
CoinbaseMessage	9	ByteArray	Optional	A message and some data for a coinbase tra
NOP_1x	1x		Optional	Values that will be ignored by anyone parsi

Scripting changes

In Bitcoin transactions version 1, checking of signatures is performed by various opcodes. The OP_CHECKSIG, OP_CHECKMULTISIG and their equivalents that immediately VERIFY. These are used to validate the cryptographic proofs that users have to provide in order to spend outputs.

We additionally have some hashing-types in like SIGHASH_SINGLE that all specify slightly different subsections of what part of a transaction will be hashed in order to be signed.

For transactions with version 4 we calculate a sha256 hash for signing an individual input based on the following content;

1. If the hash-type is 0 or 1 we hash the tx-id of the transaction. For other hash types we selectively ignore parts of the transaction exactly like it has always worked. With the caveat that we never serialize any signatures.
2. the TxId of the transaction we are spending in this input.
3. the index of output of the transaction we are spending in this input.
4. the input script we are signing (without the signature, naturally).
5. the amount, as a var-int.
6. the hash-type as a var-int.

Serialization order

To keep in line with the name Flexible Transactions, there is very little requirement to have a specific order. The only exception is cases where there are optional values and reordering would make unclear what is meant.

For this reason the TxInPrevHash always has to be the first token to start a new input. This is because the TxPrevIndex is optional. The tokens TxRelativeTimeLock and TxRelativeBlockLock are part of the input and similarly have to be set after the TxInPrevHash they belong to.

Similarly, the TxInputStackItem always has to be the first and can be followed by a number of TxInputStackItemContinued items.

At a larger scope we define 3 sections of a transaction.

Segment	Tags	Description
Transaction	all not elsewhere used	This section is used to make the TxId
Signatures	TxInputStackItem, TxInputStackItemContinued	The input-proofs
TxEnd	TxEnd	

The TxId is calculated by taking the serialized transaction without the Signatures and the TxEnd and hashing that.

TxEnd is there to allow a parser to know when one transaction in a stream has ended, allowing the next to be parsed.

Block-malleability

The effect of leaving the signatures out of the calculation of the transaction-id implies that the signatures are also not used for the calculation of the merkle tree. This means that changes in signatures would not be detectable and open an attack vector.

For this reason the merkle tree is extended to include (append) the hash of the v4 transactions. The merkle tree will continue to have all the transactions' tx-ids but appended to that are the v4 hashes that include the signatures as well. Specifically the hash is taken over a data-blob that is built up from:

1. the tx-id
2. The entire bytearray that makes up all of the transactions signatures. This is a serialization of all of the signature tokens, so the TxInputStackItem and TxInputStackItemContinued in the order based on the inputs they are associated with.

Future extensibility

The NOP_1x wildcard used in the table explaining tokens is actually a list of 10 values that currently are specified as NOP (no-operation) tags.

Any implementation that supports the v4 transaction format should ignore this field in a transaction. Interpreting and using the transaction as if that field was not present at all.

Future software may use these fields to decorate a transaction with additional data or features. Transaction generating software should not trivially use these tokens for their own usage without cooperation and communication with the rest of the Bitcoin ecosystem as miners certainly have the option to reject transactions that use unknown-to-them tokens.

The amount of tokens that can be added after number 19 is practically unlimited and they are currently specified to not be allowed in any transaction and the transaction will be rejected if they are present. In the future a protocol upgrade may chance that and specify meaning for any token not yet specified here. Future upgrades should thus be quite a lot smoother because there is no change in concepts or in format. Just new data.

Backwards compatibility

Fully validating older clients will not be able to understand or validate version 4 transactions and will need to be updated to restore that ability.

SPV (simple payment validation) wallets need to be updated to receive or create the new transaction type.

This BIP introduces a new transaction format without changing or deprecating the existing one or any of its practices. Therefore it is backwards compatible for any existing data or parsing-code.

Reference Implementation

Bitcoin Classic includes an implementation that is following this spec. The spec-author rejects the notion of reference implementation. The specification is always authoritative, the implementation is not.

The official spec can be found at; https://github.com/bitcoinclassic/documentation/blob/master/spec/transactionv4.md

Deployment

To be determined

References

1 CMF

Copyright