```
BIP: 239
Layer: Applications
Title: Transaction Extended Format (TEF)
Author:
    Simon Ordish (@ordishs)
    Siggi Oskarsson (@icellan)
Comments-Summary: No comments yet.
Comments-URI: -
Status: Proposal
Type: Standards Track
Created: 2022-11-09
```

Abstract

Regular Bitcoin transactions do not contain all the data that is needed to verify that the signatures in the transactions are valid. To sign an input of a Bitcoin transaction, the signer needs to know the transaction ID, output index, output satoshis and the locking script of the input transaction. When sending a Bitcoin transaction to a node, only the previous transaction ID and the output index are part of the serialized transaction, the node will look up the locking script and output amount of the input transaction.

We propose an Extended Format (EF) for a Bitcoin transaction, that includes the locking script and the amount in satoshis of all inputs of the transaction. This allows a broadcast service to validate all aspects of a transaction without having to contact a node or an indexer for the utxos of the inputs of a transaction, speeding up the validation.

Copyright

Motivation

Verifying that a transaction is valid, including all signatures, is not possible at the moment without getting the unspent transaction outputs (utxos) from the transactions that are used as inputs from a Bitcoin node (or a Bitcoin indexer). This lookup of the utxos always happens inside a Bitcoin node when validating a transaction, but for a broadcast service to be able to fully validate a transaction (including the fee being paid) it also needs to look up the utxos being spent, which complicates scalability, since this lookup needs to happen on a node (via RPC), that might be too busy to react within an acceptable time frame.

A broadcast service would be able to validate a transaction almost in full if the sender would also send the missing data (previous locking scripts and satoshi outputs) from the utxos being used in the transaction. When creating a new transaction, the previous locking scripts and satoshi outputs are needed to be able to properly sign the transaction, so the missing data is available at the time of the transaction creation. Serializing the transaction to Extended Format, instead of the standard format, is at the point of creating the transaction no extra work, but does make it much easier for a broadcast service to validate the transaction when being received, before sending the transaction to a node.

The main motivation for this proposal is therefore scalability. When incoming transactions contain all the data that is needed to validate them, without having to contact an external service for missing data, the broadcast service becomes much more scalable.

Specification

Current Transaction format:

Field	Description	Size
Version no	currently 2	4 bytes
In-counter	positive integer VI = [[VarInt]]	1 - 9 bytes
list of inputs	Transaction Input Structure	qty with variable length per input
Out-counter	positive integer VI = [[VarInt]]	1 - 9 bytes
list of outputs	Transaction Output Structure	qty with variable length per output
nLocktime	if non-zero and sequence numbers are < 0xFFFFFFFF: block height or timestamp when transaction is final	4 bytes

The Extended Format adds a marker to the transaction format:

Field	Description	Size
Version no	currently 2	4 bytes
EF marker	**marker for extended format**	**0000000000EF**
In-counter	positive integer VI = [[VarInt]]	1 - 9 bytes
list of inputs	**Extended Format** transaction Input Structure	qty with variable length per input
Out-counter	positive integer VI = [[VarInt]]	1 - 9 bytes
list of outputs	Transaction Output Structure	qty with variable length per output
nLocktime	if non-zero and sequence numbers are < 0xFFFFFFFF: block height or timestamp when transaction is final	4 bytes

The Extended Format marker allows a library that supports the format to recognize that it is dealing with a transaction in extended format, while a library that does not support extended format will read the transaction as having 0 inputs, 0 outputs and a future nLock time. This has been done to minimize the possible problems a legacy library will have when reading the extended format. It can in no way be recognized as a valid transaction.

The input structure is the only additional thing that is changed in the Extended Format. The current input structure looks like this:

Field	Description	Size
Previous Transaction hash	TXID of the transaction the output was created in	32 bytes
Previous Txout-index	Index of the output (Non negative integer)	4 bytes
Txin-script length	Non negative integer VI = VarInt	1 - 9 bytes

Field	Description	Size
Txin-script / scriptSig	Script	-many bytes
Sequence_no	Used to iterate inputs inside a payment channel. Input is final when nSequence = 0xFFFFFFFF	4 bytes

In the Extended Format, we extend the input structure to include the previous locking script and satoshi outputs:

Field	Description	Size
Previous Transaction hash	TXID of the transaction the output was created in	32 bytes
Previous Txout-index	Index of the output (Non negative integer)	4 bytes
Txin-script length	Non negative integer VI = VarInt	1 - 9 bytes
Txin-script / scriptSig	Script	-many bytes
Sequence_no	Used to iterate inputs inside a payment channel. Input is final when nSequence = 0xFFFFFFFF	4 bytes
Previous TX satoshi output	**Output value in satoshis of previous input**	**4 bytes**
Previous TX script length	**Non negative integer VI = VarInt**	**1 - 9 bytes**
Previous TX locking script	**Script**	**<script length>-many bytes**

Backward compatibility

The Extended Format is not backwards compatible, but has been designed in such a way that existing software should not read a transaction in Extend Format as a valid (partial) transaction. The Extended Format header (0000000000EF) will be read as an empty transaction with a future nLock time in a library that does not support the Extended Format.

Implementation

The Extended Format has been implemented in go-bt and a standalone JavaScript library bitcoin-ef.