```
BIP: 18
Layer: Consensus (soft fork)
Title: hashScriptCheck
Author: Luke Dashjr <luke+bip17@dashjr.org>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0018
Status: Proposed
Type: Standards Track
Created: 2012-01-27
License: BSD-2-Clause
```

Abstract

This BIP modifies the basic format of transaction inputs and outputs, replacing the current scriptSig and scriptPubKey (scripts executed to validate a transaction) with new contents: dataSig, scriptCheck, and hashScriptCheck.

Copyright

Motivation

The purpose of pay-to-script-hash is to move the responsibility for supplying the conditions to redeem a transaction from the sender of the funds to the redeemer.

The benefit is allowing a sender to fund any arbitrary transaction, no matter how complicated, using a fixed-length 20-byte hash that is short enough to scan from a QR code or easily copied and pasted.

Specification

scriptSig and scriptPubKey are hereby deemed to be deprecated. Bitcoin-compatible clients MUST still continue to support them for compatibility, but it should not be used for any new transaction types. Services and people which send Bitcoins SHOULD continue to support old pubkey-based addresses for the time being. Services and people which receive Bitcoins MAY continue to generate and use old pubkey-based addresses.

To replace these, there are 3 new elements:

- dataSig is included in place of scriptSig in transaction inputs, and contains multiple serialized data elements
- scriptCheck is the final element of dataSig, and is executed with the preceding dataSig elements preloaded onto the stack (the element immediately before scriptCheck is the top of the stack)
- hashScriptCheck is included in place of scriptPubKey in transaction outputs, to specify the hash of the scriptCheck allowed to redeem it

dataSig is to be encoded the same as a push-only script.

hashScriptCheck must be encoded exactly so:

```
0xa9 0x14 (20-byte-hash-value) 0x87
```

This can be interpreted by legacy (pre-BIP 18) clients as the following script:

```
OP_HASH160 [20-byte-hash-value] OP_EQUAL
```

If this template is not matched exactly OR the transaction is in a block with a timestamp before the hashScriptCheck activation date, validation MUST proceed in backward-compatibility mode, using scriptSig+scriptPubKey rather than dataSig+scriptCheck+hashScriptCheck.

A hashScriptCheck-compliant input is valid only if:

- dataSig MUST NOT contain any operations other than "push data" (it is data, not a script; no mixing scriptSig with hashScriptCheck)
- scriptCheck MUST hash (using Bitcoin's Hash160 algorithm) to the output's hashScriptCheck.
- scriptCheck MUST be executed with the dataSig-based stack specified above (ie, not including scriptCheck itself) to perform validation (this does not imply clients are required to validate transactions).
- scriptCheck must not abort, and must leave a true value on the top of the stack. This is the current behaviour for scriptSig+scriptPubKey.

The new scriptCheck SHOULD be checked against "standard transaction" templates by miners.

For example, the hashScriptCheck and corresponding dataSig for a one-signature-required transaction is:

```
   scriptCheck: [pubkey] OP_CHECKSIG
dataSig: [signature] {[pubkey] OP_CHECKSIG}
hashScriptCheck: [20-byte-hash of {[pubkey] OP_CHECKSIG}]
```

Signature operation limits for scriptCheck

Signature operations in scriptCheck do not follow the same rules previously applied to scriptSig and scriptPubKey. Instead, they shall contribute to the maximum number allowed per block (20,000) as follows:

1. OP_CHECKSIG and OP_CHECKSIGVERIFY count as 1 signature operation, whether or not they are evaluated.
2. OP_CHECKMULTISIG and OP_CHECKMULTISIGVERIFY immediately preceded by OP_1 through OP_16 are counted as 1 to 16 signature operation, whether or not they are evaluated.
3. All other OP_CHECKMULTISIG and OP_CHECKMULTISIGVERIFY are counted as 20 signature operations.

Examples:

+3 signature operations:

```
2 [pubkey1] [pubkey2] [pubkey3] 3 OP_CHECKMULTISIG
```

+22 signature operations

```
OP_CHECKSIG OP_IF OP_CHECKSIGVERIFY OP_ELSE OP_CHECKMULTISIGVERIFY OP_ENDIF
```

Rationale

This BIP replaces BIPs 12 and 17, which propose extensions to the Script system to allow scriptPubKey to outsource its verification. It also replaces BIP 16, which is identical in terms of protocol, but suggests a specific implementation and does not deprecate scriptPubKey to maintain protocol consistency.

The Motivation for this BIP (and BIP 13, the pay-to-script-hash address type) is somewhat controversial; several people feel that it is unnecessary, and complex/multisignature transaction types should be supported by simply giving the sender the complete {serialized script}. The author believes that this BIP will minimize the changes needed to all of the supporting infrastructure that has already been created to send funds to a base58-encoded-20-byte bitcoin addresses, allowing merchants and exchanges and other software to start supporting multisignature transactions sooner.

The signature operation counting rules are intended to be easy and quick to implement by statically scanning scriptCheck. Bitcoin imposes a maximum-number-of-signature-operations per block to prevent denial-of-service attacks on miners. If there was no limit, a rogue miner might broadcast a block that required hundreds of thousands of ECDSA signature operations to validate, and it might be able to get a head start computing the next block while the rest of the network worked to validate the current one.

There is a 1-confirmation attack on old implementations, but it is expensive and difficult in practice. The attack is:

1. Attacker creates a pay-to-script-hash transaction that is valid when interpreted as scriptPubKey, but contains an invalid scriptCheck, and sends themselves some coins using it.
2. Attacker also creates a standard transaction that spends the pay-to-script transaction, and pays the victim who is running old software.
3. Attacker mines a block that contains both transactions.

If the victim accepts the 1-confirmation payment, then the attacker wins because both transactions will be invalidated when the rest of the network overwrites the attacker's invalid block.

The attack is expensive because it requires the attacker create a block that they know will be invalidated by the rest of the network. It is difficult because creating

blocks is difficult and users should not accept 1-confirmation transactions for higher-value transactions.

Backwards Compatibility

hashScriptCheck transactions are non-standard to old implementations, which will (typically) not relay them nor include them in blocks.

Old implementations will validate that scriptCheck's hash value matches when they validate blocks created by software that fully support this BIP, but will do no other validation.

Avoiding a block-chain split by malicious pay-to-script transactions requires careful handling of one case:

- A pay-to-script-hash transaction that is invalid for new clients/miners but valid for old clients/miners.

To gracefully upgrade and ensure no long-lasting block-chain split occurs, more than 50% of miners must support full validation of the new transaction type and must switch from the old validation rules to the new rules at the same time.

To judge whether or not more than 50% of hashing power supports this BIP, miners are asked to upgrade their software and put the string "/P2SH/" in the input of the coinbase transaction for blocks that they create.

At 00:00:00 UTC on 15 Mar 2012, the block-chain will be examined to determine the number of blocks supporting pay-to-script-hash for the previous 7 days. If 550 or more contain "/P2SH/" in their coinbase, then all blocks with timestamps after 00:00:00 UTC on 1 Apr 2012 shall have their pay-to-script-hash transactions fully validated. Approximately 1,000 blocks are created in a week; 550 should, therefore, be approximately 55% of the network supporting the new feature.

If a majority of hashing power does not support the new validation rules, then rollout will be postponed (or rejected if it becomes clear that a majority will never be achieved).

Forwards Compatibility

The first two bytes of hashScriptCheck specify the hash algorithm and length used to verify scriptCheck. This BIP only allows Bitcoin's Hash160 algorithm, but leaves open the possibility of a future BIP implementing others.

Reference Implementation

https://github.com/gavinandresen/bitcoin-git/tree/pay_to_script_hash

See Also

- The Address format for Pay to Script Hash BIP

- BIP 16 - Pay to Script Hash (aka "/P2SH/")
- M-of-N Multisignature Transactions BIP 11