```
BIP: 16
Layer: Consensus (soft fork)
Title: Pay to Script Hash
Author: Gavin Andresen <gavinandresen@gmail.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0016
Status: Final
Type: Standards Track
Created: 2012-01-03
```

Abstract

This BIP describes a new "standard" transaction type for the Bitcoin scripting system, and defines additional validation rules that apply only to the new transactions.

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

A new standard transaction type that is relayed and included in mined blocks is defined:

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

[20-byte-hash-value] shall be the push-20-bytes-onto-the-stack opcode (0x14) followed by exactly 20 bytes.

This new transaction type is redeemed by a standard scriptSig:

```
...signatures... {serialized script}
```

Transactions that redeem these pay-to-script outpoints are only considered standard if the *serialized script* - also referred to as the *redeemScript* - is, itself, one of the other standard transaction types.

The rules for validating these outpoints when relaying transactions or considering them for inclusion in a new block are as follows:

1. Validation fails if there are any operations other than "push data" operations in the scriptSig.

2. Normal validation is done: an initial stack is created from the signatures and {serialized script}, and the hash of the script is computed and validation fails immediately if it does not match the hash in the outpoint.
3. {serialized script} is popped off the initial stack, and the transaction is validated again using the popped stack and the deserialized script as the scriptPubKey.

These new rules should only be applied when validating transactions in blocks with timestamps >= 1333238400 (Apr 1 2012) [1]. There are transactions earlier than 1333238400 in the block chain that fail these new validation rules. [2]. Older transactions must be validated under the old rules. (see the Backwards Compatibility section for details).

For example, the scriptPubKey and corresponding scriptSig for a one-signature-required transaction is:

```
   scriptSig: [signature] {[pubkey] OP_CHECKSIG}
scriptPubKey: OP_HASH160 [20-byte-hash of {[pubkey] OP_CHECKSIG} ] OP_EQUAL
```

Signature operations in the {serialized script} 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 BIP 12, which proposed a new Script opcode ("OP_EVAL") to accomplish everything in this BIP and more.

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

[1] Remove -bip16 and -paytoscripthashtime command-line arguments
[2] Transaction 6a26d2ecb67f27d1fa5524763b49029d7106e91e3cc05743073461a719776192

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.

Recognizing one 'special' form of scriptPubKey and performing extra validation when it is detected is ugly. However, the consensus is that the alternatives are either uglier, are more complex to implement, and/or expand the power of the expression language in dangerous ways.

The signature operation counting rules are intended to be easy and quick to implement by statically scanning the {serialized script}. 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 as seen by old software, but invalid for new implementation, 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

These transactions are non-standard to old implementations, which will (typically) not relay them or include them in blocks.

Old implementations will validate that the {serialize script}'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.

On February 1, 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 15 Feb 2012, 00:00:00 GMT 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).

520-byte limitation on serialized script size

As a consequence of the requirement for backwards compatibility the serialized script is itself subject to the same rules as any other PUSHDATA operation, including the rule that no data greater than 520 bytes may be pushed to the stack. Thus it is not possible to spend a P2SH output if the redemption script it refers to is >520 bytes in length. For instance while the OP_CHECKMULTISIG opcode can itself accept up to 20 pubkeys, with 33-byte compressed pubkeys it is only possible to spend a P2SH output requiring a maximum of 15 pubkeys to redeem: 3 bytes + 15 pubkeys * 34 bytes/pubkey = 513 bytes.

Reference Implementation

https://gist.github.com/gavinandresen/3966071

See Also

- https://bitcointalk.org/index.php?topic=46538
- The Address format for Pay to Script Hash BIP
- M-of-N Multisignature Transactions BIP 11
- Quality Assurance test checklist

References