Super Segwit Scripts

Author	josh
Date	2019-05-18 12:33:20

Overview

Segregated Witness, better known as segwit, is a soft-fork change to the core Bitcoin (BTC) protocol implemented to provide one possible scaling solution as well as other benefits. First, the use of segwit decreases the size of transactions (sort of) and provides a new way for the Bitcoin network to scale without a hard-fork block size increase. As well, segwit fixes *transaction malleability*, a long known problem that allows the transaction ID hash to change without changing the meaning (value transfers, etc.) of a transaction. Finally, segwit helps to decrease the amount of data nodes need to store in memory by making UTXOs much smaller.

There's a lot to segwit, and each of these points can be a lengthy technical discussion. However, it can be easier to understand these benefits of segwit by understanding how these transactions compare internally to normal P2PKH transactions.

Transaction Scripts: Segwit vs. Normal

Reviewing Legacy Transaction Scripts

First, let's review how scripts work generally and how they are used to construct transactions in Bitcoin. We'll take a look at the normal or legacy Pay-To-Public-Key-Hash transaction type, one of the most common on the Bitcoin network.

When a user receives some Bitcoin at their address, what they really receive is a new "Unspent Transaction Output" or UTXO that is associated with their address. This UTXO specifies a *locking script* - a condition that must be programatically satisfied in order to spend that Bitcoin in a future transaction.

When the user wants to send funds to someone else, their wallet constructs a transaction with an *unlocking script* that satisfies the *locking script* set on the UTXO. This script essentially *proves* that the user owns the Bitcoin to the rest of network and allows the funds to be spent.

For a typical P2PHK transaction, the locking script set by the UTXO looks like this:

```
OP_DUP OP_HASH160 <Public Key Hash> OP_EQUALVERIFY OP_CHECKSIG
```

The unlocking script the wallet will use to spend the funds looks like this:

```
<Signature> <Public Key>
```

Any node on the network that validates transactions will execute the scripts and verify the sender is indeed the rightful owner of the funds.

Segwit Transaction Scripts

Now that we've reviewed traditional P2PKH scripts and validation, it's actually fairly simple to understand how segwit scripts operate. There are two big differences: the *locking script* uses a different format, and the location of the *witness* or *unlocking script* is outside of the transaction input!

First, let's look at the locking script placed on an output sent to a segwit-compatible address:

```
<version> <pubkeyhash>
```

The segwit locking script simply pushes a version number (for example, `OP_0` to the stack, followed by the public key hash. This serves two important purposes. Similar to a P2PKH transaction, it provides the hash of the address owner's public key. But more interestingly, this functionally lets nodes that are not running a segwit implementation see the UTXO as "anyone can spend", making the transaction backwards-compatible on the network.

The normal unlocking script would be in the *scriptSig*, but for segwit transactions, this field is actually left empty! Instead, an entirely new data structure called the *witness* is used. The witness contains the same data as a normal P2PKH scriptSig would though, so it's pretty easy for us to understand:

```
<Signature> <Public Key>
```

Thesegwit BIP (Bitcoin Improvement Proposal) states that this script will be verified by simply checking the signature:

```
The signature is verified as

    <signature> <pubkey> CHECKSIG
```

Segwit Scripts - Same Idea, Different Data

Although segwit in its entirety is a complex topic, it's fairly trivial to understand the difference in how scripts are implemented and how the corresponding transactions are validated. With every Bitcoin transaction, a user owns some UTXOs with a locking condition placed on them. When this user wants to spend some Bitcoin, they create a new transaction with a signature that satisfies that script, "unlocking" the currency by proving they are the rightful owner.

With P2PKH legacy transactions, this is done with a signature in the scriptSig field. With segwit, this unlocking signature is moved to a separate ("segregated") data structure called the witness. The internal mechanisms of the Bitcoin implementation validates the signature with an `OP_CHECKSIG` similar to a normal transaction. And interestingly, the original locking condition allows backwards compatibility with old nodes on the network.