2.4-Addresses.md

# Addresses

An address is typically used as a recipient to a Bitcoin transaction. The address, in fact, is just an encoded version of a specific type of `scriptPubKey`. As we learned earlier, the `scriptPubKey` locks the funds until a valid `scriptSig` is provided, which, when executed together, will result in true. There are a few types of addresses, and they are encoded for convenience and use within wallet applications. These types are:

1. Pay to Pubkey Hash (P2PKH)
2. Pay to Script Hash (P2SH)
3. Bech32

In the following sections, we'll describe how to create each type of address from scratch. First a quick primer on the cryptography used in Bitcoin.

## Bitcoin Cryptography

Bitcoin uses Elliptic Curve Cryptography for signing transactions. Elliptic Curve is a form of Public-key or Asymmetric Cryptography. Users store their private keys, which allow them to sign transactions to prove they own and can send their funds. With these private keys, anyone could spend the funds, so it is very important to keep them secret. When coins are sent to a user, they should be sent to a script that requires a valid signature by including the public key provided by the intended recipient.

First, the user selects a random private key. The private keys can be any 256 bit number from `0x1` to `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364140`. This is the largest 256 bit number within the finite field of the `secp256k1` elliptic curve used by Bitcoin. This is an extraordinarily large range, in fact its practically incomprehensible the number of possibilites. This is important, because you wouldn't want someone else to be able to guess your private key and gain access to all of your funds. There are approximately 2^256 or 1.6E77 possible private keys. The number of atoms in the visible universe is estimated at around 10E80, see Observable Universe.

### Key Pairs

Let's generate a private key using SHA256 (since it will always give us a 256 bit number:

```
$ sha256 "Don't use this private key it is no longer safe"
84cf749059a129fb57a6070411234a4fe58e7f04c959b66fa7c8b2d2dd609749
```

Next, we need to calculate the corresponding public key using the `secp256k` elliptic curve. OpenSSL can be used for this:

```
$ pubkey 84cf749059a129fb57a6070411234a4fe58e7f04c959b66fa7c8b2d2dd609749 –u
04a097026e876544a0e40f9ca836435560af4470e161bf60c23465dcb3151c947d1cbe052875211972107e25fca8dd939f1c6e749a
```

The `0x04` designates this as an uncompressed public key, where the next 32 bytes are the x value and the final 32 bytes are the y value of the point on the elliptic curve, respectfully.

**Compressed public keys**

Most wallets and nodes implement compressed public key as a default format because it is half as big as an uncompressed key, saving blockchain space. To convert from an uncompressed public key to a compressed public key, you can omit the y value because the y value can be solved for using the equation of the elliptic curve: $y^2 = x^3 + 7$, given x. Since the equation solves for $y^2$, y could be either positive or negative. So, `0x02` is prepended for positive y values, and `0x03` is prepended for negative ones. If the y coordinate is even, then it corresponds to a positive number. If odd, then it is negative. Since the y value ends in `0xee`, which is even, and therefore positive, the compressed version of the public key becomes:

`03a097026e876544a0e40f9ca836435560af4470e161bf60c23465dcb3151c947d`

You can also omit the `-u` flag in the above command.

We will use this same public key to create different types of addresses.

## Pay to Pubkey Hash (P2PKH)

This address is a standard transaction type where the sender locks funds with the hash of another user's public key. A hash is shorter than a public key (20 bytes instead of 33), so it is used mainly for convenience. In order to redeem the funds, the receiver must provide both their public key and a valid transaction signature. First we calculate the HASH160 of the public key, which is just the RIPEMD160(SHA256(compressed-key)):

```
$ hash160 03a097026e876544a0e40f9ca836435560af4470e161bf60c23465dcb3151c947d
828a58e0b8731f81e9f0f4e789a6ddeef62ef108
```

Next, we construct the address. In order for the wallet software to know how to construct the `scriptPubKey` corresponding to the address, we use a version byte prefix of `0x00` to denote a `P2PKH` script (since addresses are not used in the blockchain itself, only scripts):

`00828a58e0b8731f81e9f0f4e789a6ddeef62ef108`

Then, we encode this using Base58Check encoding. This is base58 with a checksum, which helps prevent a typo from sending a transaction to the wrong address.

```
$ bs58 -c 00828a58e0b8731f81e9f0f4e789a6ddeef62ef108
1CuEaUAvhm8D9SkwEVLgqvFShHAxKmbW19
```

## Pay to Script Hash (P2SH)

A more commonly used address type is the pay-to-script-hash, where instead of hashing a public key, you hash a script. A commonly used script is a segregated witness script called pay-to-witness-pubkey-hash, which is a hash of a public key. The script takes the form `OP_0 0x14 {pubKey hash}`. `OP_0` indicates a witness script, and `0x14` is the number of bytes in the pubkey hash. Our script becomes `0014828a58e0b8731f81e9f0f4e789a6ddeef62ef108`.

Then we base58check encode it with a prefix of `0x05` to denote that this is a script hash.

```
$ hash160 0014828a58e0b8731f81e9f0f4e789a6ddeef62ef108
bd19b54ad03e35878a5b6d4c39c4888881422629

$ bs58 -c 05bd19b54ad03e35878a5b6d4c39c4888881422629
3JvtS2yYE9Uoy6cjJrB9t7PEeR7nYHodgH
```