# Contents

# Part I

# The Basics

# Overview

In this part, we explain a number of basic concepts that will be referred to in later chapters. For a more thorough and structured introduction to Bitcoin, consider reading *Mastering Bitcoin* by Andreas M. Antonopoulos,[2] or *Grokking Bitcoin* by Kalle Rosenbaum.[3] However, these books aren't required reading to follow along with this book.

In chapter 1, we explain how a Bitcoin address isn't something that exists on the blockchain, but rather is a convention used by wallet software to communicate where coins must be sent. We also explain how these addresses are encoded using base58, and more recently with bech32.

In chapter 2, we explain how, the very first time your Bitcoin node starts up, it finds peers to communicate with. You'll also get a primer on Tor.

Chapter 3 explains the 2017 SegWit soft fork and talks about how it increased the block size and paved the way toward the Lightning network by solving transaction malleability.

Finally, chapter 4 explains what libraries are, how they cause problems, and what happened with OpenSSL in particular.

[2]https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/

[3]https://www.manning.com/books/grokking-bitcoin

## Reading Hints

Each chapter contains a QR code that takes you to the corresponding podcast episode and its show notes. The episode number is shown below the QR code. You can also find the episode in your favorite podcasting application by searching for *Bitcoin, Explained.* Or, play it in your browser.[4]

There is a tiny QR code displayed next to each URL. These go through btcwip.com in order to keep them small. We won't track you.

Throughout the text, there are many references to the Bitcoin Core node software and its built-in wallet. Appendix B has some screenshots of it and shows a typical workflow for sending and receiving bitcoin.

[4]https://nadobtc.libsyn.com

# Chapter 1

# Bitcoin Addresses

Ep. 28

Bitcoin addresses aren't part of the Bitcoin blockchain; rather, they're conventions used by Bitcoin (wallet) software to communicate where coins must be sent to: either a public key (P2PK), a public key hash (P2PKH), a script hash (P2SH), a witness public key hash (P2WPKH), or a witness script hash (P2WSH). Addresses also include some metadata about the address type itself.

Bitcoin addresses communicate these payment options using their own numeric systems, and this chapter will break down what these different systems mean. It'll also delve into some of the benefits of using Bitcoin addresses in general and bech32 addresses specifically. We explain how the first version of bech32 addresses included a (relatively harmless) bug, and how it was fixed. We finish the chapter with some quantum talk.

## Some History

When you send bitcoin to someone, you're creating a transaction that has several inputs and at least one output. The output specifies who can spend it by putting a constraint on

it — a fancier term for constraint is encumbrance.[1]

The most trivial encumbrance is that anybody can spend the coin. That's not a good idea, because it'll be stolen very quickly. So in the early days, most coins on the blockchain were encumbered in one of two ways: Pay-to-Public-Key (P2PK) and Pay-to-Public-Key-Hash (P2PKH). The first can be read as "only the holder of the private key corresponding to the public key X may spend this coin," and the second as "only the holder of the private key corresponding to a (secret) public key, which hashes to X, may spend this coin."

Back then, it was possible to send bitcoin to people's IP addresses,[2] but this feature was dropped in 2012. When this was possible, you could connect to someone's IP address and ask for a public key, and the person would give you their public key.[3] Your wallet would then create a new coin encumbered with a P2PK script.

Today, this workflow might seem strange,[4] but it matches the then-common pattern of peer-to-peer apps like Napster or Kazaa, where you'd connect directly to other people and download things from them. Nowadays, you probably don't know the IP addresses of your friends, and they might even change all the time when they're on mobile devices. Although you can instruct your Bitcoin node to specifically connect to a friend's node, it typically just connects to random peers (see chapter 2).

[1]https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch05.html#tx_script

[2]https://en.bitcoin.it/wiki/IP_transaction

[3]For the curious code archaeologist: On the sender node there was a UI dialog that prompted for an amount and IP address. The function `StartTransfer()` created a blank cheque transaction, into which `checkorder` on the recipient node would insert a P2PK script (as the `scriptPubKey`). `OnReply2()` would then insert the amount, sign the transaction, return it to the recipient and also broadcast it. https://github.com/bitcoin/bitcoin/blob/v0.1.5/main.cpp#L2004-L2042

[4]And unsafe, as Satoshi acknowledged: https://bitcointalk.org/index.php?topic=158.msg1322#msg1322

The more common way of doing transactions is similar to how bank transfers work. Someone provides you with an address and you send coins to it, just as you'd send money to a bank account number. This initially always used P2PKH, as we'll explain below.

Instead of sending a transaction directly to the recipient, it makes its way through all the nodes on the network, eventually to be seen by a miner node, which includes it in a block. Your counterparty may see the transaction as their node receives it from one of its peers, or they'll see it once they receive the block it's in.

A third way of doing transactions is to mine bitcoin, which involves sending the block reward to yourself. In the beginning, Bitcoin had a piece of mining software built into the software, so if you downloaded the Bitcoin software, it would just start mining. It would then send coins to your own wallet, so there was no need to communicate an address. Those coins were encumbered with P2PK.[5]

## So What Is an Address?

An address is a convenient way to communicate which script needs to go on the blockchain. As we said above, the purpose of this script is to constrain the coin so that only the recipient can spend it.[6] The address itself doesn't exist on the blockchain. It doesn't even contain the full script.

[5]Why did Satoshi's initial version support both P2PK and P2PKH? We're not sure. The P2PK way of paying someone was only ever really used for paying to an IP address, and for the miner, the block reward. Neither needed human interaction. In scenarios that did involve human interaction, P2PKH was used. When using an address, P2PKH, and not P2PK, is implied. Automated systems don't require the concept of an address, since they can just as well handle script, and so there's no such thing as a P2PK address.

[6]So far, the analogy to bank accounts holds, but we'll learn in chapter 10 that scripts can be far more powerful than just functioning as buckets to hold money for their owners.

Of the two main types of scripts in use back in the day, addresses were only used for Pay-to-Public-Key-Hash (P2PKH). When a wallet sees this address, it produces a script for the Bitcoin blockchain, which requires that the person spending it has the public key belonging to the hash (chapter 10 contains the actual script). Only the hash is published, so the public key remains secret until the recipient spends the coins.

An address starts with the number 1, followed by the hash of the public key. It's encoded using something called base58. Here's an example: `1HLoFgMiDL3hvACAfbkDUjcP9r9veUcqAF`

## Base Systems Explained

To understand what base58 is, it's important to first understand more about base systems in general.

With base10, think about your hand. You have 10 fingers. So if you want to, for example, express the number 115 (1, 1, 5), you can make three gestures with your hands by showing 1, 1, and 5. That's also how you write down numbers, which — since the invention of clay tablets and paper — is more convenient than using fingers. So base10 is a decimal system that uses 10 different symbols, in various combinations, to represent any number (integer).

However, there have been — and still are — different bases. For example, the Babylonians[7] used base60. And to read machine code, typically you'd use hexadecimal,[8] which is base16 — 0 to 9, and then A to F. Meanwhile, computers tend to use base2 internally — a binary number system — because transistors are either on or off. This translates to

[7]https://blogs.scientificamerican.com/roots-of-unity/ancient-babylonian-number-system-had-no-zero/

[8]https://en.wikipedia.org/wiki/Hexadecimal

using two digits, either 0 or 1, to do everything, and you can express any number that way.

Satoshi introduced[9] base58, which uses 58 different symbols: 0 through 9, and then most of the alphabet in both lowercase and uppercase. But there are some letters and numbers that are skipped because they're ambiguous and users could easily mistake them for the wrong one — for example, the number 0 and the uppercase letter O, and capital I and lowercase l.

Have you ever seen email source code for an attachment or similar? There are a lot of weird characters. That's base64, and base58 is based on that. But base64 includes characters like underscores, plus, equals, and slash. These are omitted in base58 to make visual inspection easier and to behave nicely as part of a URL.

## Base58 and the Pay-to-Public-Key-Hash

So how does this relate to P2PKH? Well, the address is expressed as a 1, followed by the public key hash, which is expressed in base58.

That's the information you send to somebody else when you want them to send you bitcoin. You could also just send them 0x00,[10] and then the public key. And maybe they'd be able to interpret that, but probably not.

In theory, you could send somebody the Bitcoin script in hexadecimal, which is the format used on the blockchain, because that's just binary information. The blockchain has this script that says, "If the person has the right public key hash and the public key belonging to this public key hash,

[9]https://tools.ietf.org/id/draft-msporny-base58-01.html

[10]A pair of hexadecimal digits, prefixed by 0x, is often used to denote bytes, which contain $16 \times 16 = 256$ bits, so this represents one byte with the value 0.

then you can spend it." To learn more about how Bitcoin scripts work, refer to chapter 10.

But even with all these options, the convention is that you use this standardized address format, which explains why all traditional Bitcoin addresses start with a 1, and why they're all roughly the same length.

In addition to using base58 for sending a Bitcoin address, you can also use it to communicate a private key. In such a scenario, the leading symbol is a 5, which represents 128. That's then followed by the private key.

In the past, users had paper wallets they could print. And if they were generated securely without a back door, then on one side of the piece of paper would be something starting with a 1, and on the other side of the paper would be something starting with a 5. And then it specified that only the Bitcoin address should be shown, but the private key shouldn't be shared.

There are also addresses that begin with a 3, which is for coins encumbered by the hash of a script, rather than the hash of a public key. We'll cover Pay-to-Script-Hash (P2SH) in chapter 10. Usually these are multi-signature addresses, but they could also be SegWit addresses.[11]

Although base58 addresses worked fine, there was room for improvement. And this came in the form of bech32.

[11]As explained in chapter 3, SegWit typically uses bech32 addresses. But it took a long time for all wallets and exchanges to support sending to bech32 addresses. To still take advantage of some of SegWit's benefits, an address type that looks like regular P2SH to the sender was introduced, but it contains SegWit magic under the hood. This is called a P2SH-P2WPKH address: https://bitcoincore.org/en/segwit_wallet_dev/

## Along Came Bech32

In March of 2017, Pieter Wuille spoke about a new address format,[12] bech32, and it's been used since SegWit arrived on the scene. As the name suggests, it's a base32 system, which means you have almost all the letters, and almost all the numbers, minus some ambiguous characters that you don't want to have because they look too much like other numbers or letters.

One of the biggest differences between bech32 and base58 is that there isn't a mixture of uppercase and lowercase letters. Instead, each letter is only in there once — either in all uppercase or all lowercase — which makes reading things out loud much easier. The precise mapping of which letter or number corresponds to which value is, like in base58, fixed but arbitrary: The fact that P means 0 and Q means 1 has no deeper meaning.

Table 1.1: Bech32 mapping. E.g. `q` means zero, `3` means 17 (1 + 16)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| +0 | q | p | z | r | y | 9 | x | 8 |
| +8 | g | f | 2 | t | v | d | w | 0 |
| +16 | s | 3 | j | n | 5 | 4 | k | h |
| +24 | c | e | 6 | m | u | a | 7 | l |

A bech32[13] address consists of two parts separated by 1, e.g. `bc1q9kdcd08adkhg35r4g6nwu8ae4nkmsgp9vy00gf`.

The first part is intentionally human readable, e.g. "bc" (Bitcoin) or "lnbc" (the Lightning network on Bitcoin). The values represented by "b," "c," etc. have no meaning. Rather, they're there so humans can recognize, "OK, if the address starts with bc, then it refers to Bitcoin as the currency."

[12]https://www.youtube.com/watch?v=NqiN9VFE4CU

[13]Bech32 spec (BIP 173): https://en.bitcoin.it/wiki/BIP_0173

However, wallets will look for the presence of these values as a confidence check, and it's included in the checksum.

The 1 is just a separator with no value. And if you look at the 32 numbers, 1 isn't included — it means "skip this."

The second part starts with the SegWit version number. Version 0 is represented with Q (bc1q...) — see chapter 3. Version 1 is what we call Taproot (see part V), as it's represented with "P" (bc1p...). For version 0 SegWit, the version number is followed by either 20 bytes or 32 bytes, which means it's either the public key hash or the script hash, respectively. And they're different lengths now, because SegWit uses the SHA-256 hash (32 bytes) of the script, rather than the RIPEMD160 hash (20 bytes) of the script.

In base58, the script hash is the same length as the public key hash. But in SegWit, they're not the same length. So by looking at how long the address is, you immediately know whether you're paying to a script or you're paying to a public key hash. As an aside, Taproot removes this length distinction, thereby slightly improving privacy.

So the new part is that there's a set of 32 characters, but otherwise, things are very similar to base58. It's again saying, "OK, here's a P2PK address." In this case, it's a Pay-to-Witness-Public-Key-Hash (P2WPKH), where witness refers to SegWit, but it's the same idea. There's a short prefix that tells both humans and the computer what the address is about, and this is followed by the hash of the public key or script.

## Thirty-Two Dimensional Darts

However, conciseness isn't the only benefit here. Another is error correction, or at least detection.

If there's a typo in an address, then in the worst case scenario, you're sending coins to the wrong hash of a public key. When the recipient tries to spend the coin, they reveal the public key, but due to the typo, its hash won't match what the blockchain demands. The coins are forever lost.

Fortunately, base58 addresses contain a checksum at the end. That way, if you make a typo, the checksum at the end of the address won't work. Your wallet will alert you to this, and it'll refuse to send the transaction (the blockchain won't protect you; only your wallet will, hopefully). But if you're really unlucky, a typo can be such that it produces a correct checksum by sheer coincidence.

Bech32 was designed in such a way to make such a disastrous coincidence extremely unlikely. In addition, it won't just tell you that there *is* a typo; it can tell you *where* the typo is. This is determined by taking all the bytes from the address and then hashing it using some sophisticated mathematical magic.[14] You can make about four typos and it'll still know where the typo is and what the real value is. If you do more than that, it won't.

To illustrate this conceptually, it's like if you have a wall and you draw a bunch of non-overlapping circles on it. The bullseye of each circle represents a correct value, whereas any other spot within the circle represents a typo. If you're a good dart player, most of the time you'll hit the bullseye, i.e. you typed the correct value. If you slightly miss the bullseye but you're still within that big circle, the value will be slightly incorrect. Error *detection* is knowing that you missed the bullseye. Error *correction* is the equivalent of moving the dart to the nearest bullseye.

The idea there is you want the circles to be as big as possible, to facilitate even the sloppiest dart thrower, but you don't want to waste too much space. Similarly, we don't want Bitcoin addresses to be hundreds of characters long. That's the kind of optimization problem mathematicians love.

[14]Math behind bech32 addresses: https://medium.com/@MeshCollider/some-of-the-math-behind-bech32-addresses-cf03c7496285

In the case of bech32, instead of a two-dimensional wall, you have to somehow imagine a 32-dimensional "wall" with 32-dimensional hyperspheres. You're hitting your keyboard, and somewhere in that 32-dimensional space, you're slightly off, but you're still inside this hypersphere, whatever that might look like. In that case, your wallet knows where the mistake is, and it prevents you from sending coins into the ether.[15]

## But... There's a Problem

In 2019 it was discovered that, if a bech32 address ends with a P, then if you accidentally add one or more Qs to it, it still will match the checksum, and you won't be told there's a typo. In turn, your software would think it's correct, you'd be sending money to the wrong address, and it would be unspendable, as we explained above.

The good news is that bech32 was only used for SegWit, and SegWit addresses were constrained — they had to be either 20 bytes or 32 bytes. So luckily, if you add another Q to a 20- or 32-byte address, then its length would be invalid. Your wallet would detect this and refuse to send coins. A similar size constraint was considered for Taproot, but thanks to the solution below, it wasn't needed. A flexible length makes future improvements to Taproot easier.

[15]Pardon the pun, but early Ethereum wallets didn't use error detection, because their address standard lacked a checksum. Although EIP 55 introduced such a checksum in 2016, not all wallets enforced it. Even in late 2017, people lost coins due to typos: https://bitcointalk.org/index.php?topic=2161699.0

## Enter Bech32m

To fix this bug, a new standard called bech32m was proposed.[16] It's actually a very simple change. It adds one extra number to the bech32 checksum math, which then makes sure no extra characters can be added without causing an invalid checksum.

The new standard is only used for Taproot and future addresses. For SegWit, nothing changes, because it's already protected by the 20- or 32-byte length constraint. At the time of writing, most wallet software supports the new bech32m standard.

## How I Learned to Stop Worrying and Love Quantum

As an aside, Pay-to-Public-Key-Hash (P2PKH) was thought to be safer against quantum attacks, because you didn't have to say which public key you had. The downside was that the hash consumed more block space — but this wasn't an issue back then, because blocks were nowhere near full.

Many people are worried that quantum computers will eventually break the security offered by Bitcoin's cryptography, allowing future quantum hackers to steal coins, potentially crashing the market if they steal millions.

The problem is that, despite widespread P2PKH use, there's 5 to 10 million BTC out there for which the public key is already known. The irony is that because so much BTC is already vulnerable to quantum theft, there's no use trying to protect the rest. Even if your coins won't be stolen, they'll be worthless from the price crash.

The (un)likeliness of such quantum troubles in the near future, as well as possible countermeasures, is explained in

[16]Bech32m spec (BIP 350) https://en.bitcoin.it/wiki/BIP_0350

two *What Bitcoin Did* podcast episodes — with physicist Stepan Snigirev[17] and mathematician Andrew Poelstra.[18]

Block space is much scarcer now, so not having to put public key hashes on precious block space would save users fees. This is why in the new Taproot soft fork (see part V), Bitcoin addresses are P2PK again.[19,20] Note that the use of Taproot addresses isn't mandatory, so if you don't agree with the above reasoning, you can simply choose to not use Taproot.

[17]https://www.whatbitcoindid.com/podcast/the-quantum-threat-to-bitcoin-with-quantum-physicist-dr-stepan-snigirev

[18]https://www.whatbitcoindid.com/podcast/andrew-poelstra-on-schnorr-taproot-graft-root-coming-to-bitcoin

[19]Full rationale in BIP 341: https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki#cite_note-2

[20]https://twitter.com/pwuille/status/1409560741489778688