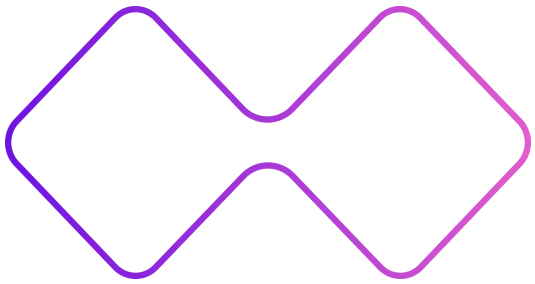


MWC GUIDE

Mimblewimble Coin // Private, Scarce, Scalable

About

The purpose of this guide is to provide an overview of the Mimblewimble protocol and Mimblewimble Coin (MWC). MWC aims to be the most potent actualization of monetary sovereignty on the planet and is the result of many innovations over decades. Being the result of many innovations; MWC is a very complex monetary product that has resulted in the combined superior characteristics of scalability, security and privacy.

Imagine: a pure proof of work private and fungible ossified cryptocurrency with 417 stock-to-flow with users running a full-node on their mobile device.

We introduce MWC in the context of good money principles, explaining why MWC evolved from bitcoin, how it complements bitcoin for those seeking to exercise their monetary sovereignty, how to use it, and what the future holds.

The overarching purpose of this overview is to enable the user to have the highest, most potent form for monetary sovereignty available to them. The individual should be able to obtain monetary sovereignty at the lowest cost possible, and should be able to flex their monetary sovereignty muscle as easily as possible.

As we explain throughout this guide, these aspects of monetary sovereignty are uniquely possible because of the privacy, scarcity and scalability provided by MWC. These attributes have never been available before and together they make up a very powerful form of money for the individual to harness.

This guide assumes some understanding of bitcoin.

Table of Contents

What is Good Money?

If we are to pursue the mission of achieving monetary sovereignty and the freedom to transact, we must identify the best, purest, most potent form of money that exists. In order to understand how to qualitatively measure different forms of money, we begin with the facets that define good, sound money. These facets are: recognizability, scarcity, censorship resistance, durability & indestructibility, extensibility, salability, portability, fungibility, privacy and divisibility.

A. Recognizability

The first attribute of good money is that it must be recognizable. It should be easy for any person or entity to authenticate that the form of money is valid. When it comes to traditional currencies such as gold or fiat, there's a hurdle to ensuring authenticity. In the case of gold, one must have access to a refinery in order to melt it down and confirm that it indeed has the properties of gold. In the case of fiat, one must be able to confirm that paper currency has not been counterfeited, and that digital units of the currency have not been tampered with.

In the case of cryptocurrency, it is much easier to confirm recognizability. Any individual can run their own node which validates transactions and confirms the authenticity of the units of account. As we will see later, different cryptocurrencies have different hurdles to be able to run a full node, which means that recognizability comes at varying costs.

B. Scarcity

Any form of good or sound money must also be scarce. In most assets, as the value of the asset increases, the powers that be will issue more of that asset in order to enrich themselves. In the stock market, this comes in the form of issuing more shares as the market capitalization increases. In fiat currencies, governments increase the supply by printing more paper currency. In all cases, this

devalues the currency at the expense of holders and in favor of those closest to the issuance.

There is a spectrum to which the supply of an asset increases relative to its base, and this spectrum is measured by the stock-to-flow ratio. Stock-to-flow is a measure of current stock relative to the annual flow, or new emission. Gold is an example of a commodity with a high stock-to-flow its supply cannot be easily increased relative to the amount outstanding; bitcoin has an even higher stock-to-flow given its fixed supply and the fact that its emission rate continues to decrease due to halvings. As we will see later, MWC has the highest stock-to-flow.

C. Censorship Resistance

A good money must be censorship-resistant so that any individual or entity can send and receive payments and store value without a central party intervening. Traditional forms of intervention range from money transmission laws and political sanctions to taxation and outright confiscation.

Bitcoin provides a form of censorship-resistance since transactions are transmitted through a wide network of full nodes which cannot be shut down by a single party such as a government or a bank. However, because all addresses on the bitcoin blockchain are public, these parties can apply pressure in order to blacklist addresses and freeze transactions, which infringes on bitcoin's censorship-resistance.

D. Durability and Indestructibility

Good money must be durable, such that it cannot be destroyed. Something like cash can be lit on fire and evaporated, which is not very durable; digital units of a cash currency are similarly vulnerable to theft or loss by the party in control. In the case of commodities such as gold, there is a high degree of durability. It's quite difficult to erode or destroy gold.

Bitcoin similarly is quite durable because it is nearly impossible to tamper with it, and copies of the ledger are stored on nodes around the world. To the degree nodes become more centralized, either due to the cost of running a node or due to the government targeting node operators, bitcoin's durability gets reduced. However, bitcoin has the network effects to remain significantly more durable and indestructible than other forms of money.

E. Extensibility

A unique aspect of cryptocurrencies is extensibility. Cryptocurrencies are open-access APIs which means that anyone anywhere can build upon them, providing monetary innovation and increasing aspects of usability. This type of openness does not exist with fiat currencies, which are controlled by central governments and banks, nor does it exist with commodities which are limited by their physical properties.

In the case of cryptocurrency, different cryptocurrencies have different levels of extensibility based on their design; some provide more scripting capabilities such that any application can be created on top of the protocol. Others are more limited, often in favor of other tradeoffs. Ethereum has a high degree of extensibility at the expense of being a form of money at all; bitcoin is a solid form of good money with decent extensibility through bitcoin script; MWC has less in the way of scripting but provides more monetary innovation through its base layer characteristics.

F. Salability

Good money must be salable which means that it can be transferred and transacted in with minimal slippage and costs. Aspects that contribute to salable money are liquidity, fungibility and freedom to transact. Fiat currency is highly salable because of how liquid it is; the same is true of gold. Bitcoin has increased

substantially in liquidity since its launch and as such it has become much easier to transact in it and to exchange it for other forms of currency.

Liquidity is a function of how many market participants are transacting and in how much size. Ultimately, the more that a form of money is available to any participants, without any barriers or gatekeepers, the more that it will be able to increase in liquidity and salability.

G. Portability

Money must be portable which means we must be able to easily transfer it. The more easily money can be transferred, the more it can be used for transacting for a wide range of people around the world. Any physical form of money, such as cash or gold, has a low degree of portability. Digital units such as USD wires and transfers are more portable, but they are subject to money transmission laws and other limitations such as transferring between countries.

Bitcoin extends the portability that comes from digital transmission by also enabling disintermediated transfers to anyone, anywhere, nearly instantaneously. Additionally, bitcoin can be transported easily due to the ability to store it on devices such as a USB or hardware wallet. However, bitcoin portability has limitations due money transmission laws and the ability for regulators to interfere with transfers through blockchain surveillance and address whitelisting/blacklisting.

H. Fungibility

Next, we have the aspect of fungibility, which is very important to any form of good money and to its liquidity. Fungibility means that money is easily interchangeable, which stems from all units of the money being equivalent. At its most basic level, gold is highly fungible because its atomic properties remain the same no matter what. Any unit of gold can be exchanged for any other unit of gold and there is no distinction between them.

Fiat currency is less fungible as there are things like serial numbers which start to make each unit unique and distinguishable from other units. Bitcoin is also less fungible because any UTXO originates from an address, and that address provides a unique provenance for each UTXO, which enables any user to distinguish some UTXOs from others. For example, if a UTXO originated from a blacklisted address, those bitcoins may not be accepted by a merchant or an exchange.

I. Privacy

Privacy is related to the aspect of fungibility. If transactions are not private, then any party can start to differentiate amongst the units based on their origin, history, and from where they originate. Fiat exchanges have an extremely low degree of privacy since every transaction is surveilled by banks and governments. They can monitor who is transacting, when they're transacting, the amount in which they're transacting, and from where they are transacting. These aspects are a substantial hindrance to a good form of money.

While bitcoin is much better than fiat due to the pseudo-anonymous nature of addresses, it is still recording all details of transactions on the public blockchain. As we've seen, this has led to blockchain analytics companies that are continually monitoring activity either for purposes of information gathering, or for purposes of money transmission laws and address blacklisting. In all cases, the availability of transaction information on the public blockchain has meant that bitcoin is not a form of private money. It offers many other advantages, but privacy is not among them.

J. Divisibility

Finally, any form of money must be divisible. It must have the ability to be broken down into small, equivalent units. This aspect is difficult for something like gold, which requires refineries to melt it down. It's more possible with fiat, which can be digitally divided into dollars and cents.

Bitcoin has even more divisibility since each unit can be divided into one-one hundred millionth of a bitcoin. This means that bitcoin can be more easily used for small transactions even as the price of bitcoin continues to increase.

However, as we noted above, not all units of bitcoin are equivalent because of the ability to distinguish between them. So, while bitcoin offers a high degree of divisibility, there is room for improvement.

Intro to MWC; Match and Exceed

We have outlined attributes of good money, and have noted how bitcoin has been a step function improvement over the previous forms of money such as fiat and gold. Now, we ask the question whether there is room for further innovation and improvement when it comes to satisfying all facets of good money. In order for a new form of money to improve upon bitcoin, it must match and exceed every aspect of good money.

The Mimblewimble whitepaper was anonymously introduced in 2016. In 2019, Mimblewimble Coin (MWC) was introduced. The initial stock was created via airdrop to Bitcoin holders. For consensus, MWC is a pure proof-of-work coin that provides substantial scalability and privacy features beyond what bitcoin has to offer. From a technical perspective it “matches and exceeds” in every area.

Gold and silver coexisted for millennia as complimentary monetary goods. In the post-Bitcoin monetary era, gold, silver and Bitcoin have coexisted as monetary goods. Just like Bitcoin is not a replacement for gold and they coexist in a new post-Bitcoin monetary era, likewise MWC is not a replacement for bitcoin. The two can and will coexist in the new economic era defined by cryptographically secure sound money.

Bitcoin has been a wonderful experiment to show what is possible for these new cryptographically secure sound money in terms of network effects and liquidity.

However, the potential for MWC to match and exceed bitcoin exists across all facets of good money. How? Let's have a look:

Good Money	BTC	MWC	Match or Exceed
Recognizability	Anyone can authenticate by running full node	More lightweight, scalable blockchain; easier to run full node	Exceed
Scarcity	Stock-to-flow of 120	Stock-to-flow of 417	Exceed
Censorship Resistance	Network of full nodes, but trackability of addresses and amounts	No addresses and amounts; very difficult for censorship	Exceed
Durability	Decentralized ledger held on nodes throughout the world	Decentralized ledger across fewer nodes right now, but with potential for more than bitcoin due to scalability	Match
Extensibility	Open-access API with bitcoin script for extensibility	Open-access API designed for monetary sovereignty extensibility	Match
Saleability	Highly salable due to liquidity	Less liquid right now but ability for more market participants in the future due to lack of censorship	Match
Portability	High degree of digital portability, but transfers may be infringed upon by laws/regulators and blockchain surveillance	Equally portable in the digital sense, and additionally immune from addresses being blacklisted and transfers frozen	Exceed
Fungibility	Decreased fungibility due to address tracking which enables differentiating between UTXOs	No addresses means that all outputs are equivalent	Exceed
Privacy	Addresses and amounts of every transaction are public	No addresses and no amounts on the public ledger	Exceed
Divisibility	Highly divisible	Highly divisible with all amounts being equivalent due to fungibility	Match

How MWC works

Now that we have a sense of where MWC stacks up in terms of a form of money, let's dig into how exactly it accomplishes this.

Confidential Transactions

MWC builds upon great work that was done in the bitcoin space. Among the innovations that MWC uses are Confidential Transactions, which means that all amounts are blinded. The sender and receiver know the amount that was transacted, and both sides sign the transaction to commit to it, but the amount is never revealed on the public blockchain.

Confidential transactions on Mimblewimble utilize Pederson Commitments. These commitments have homomorphic properties which means that we can perform calculations on encrypted values, and can verify that the computations satisfy certain properties, without ever needing to reveal the values.

A calculation is performed and verified on A and B, but A and B are never revealed.

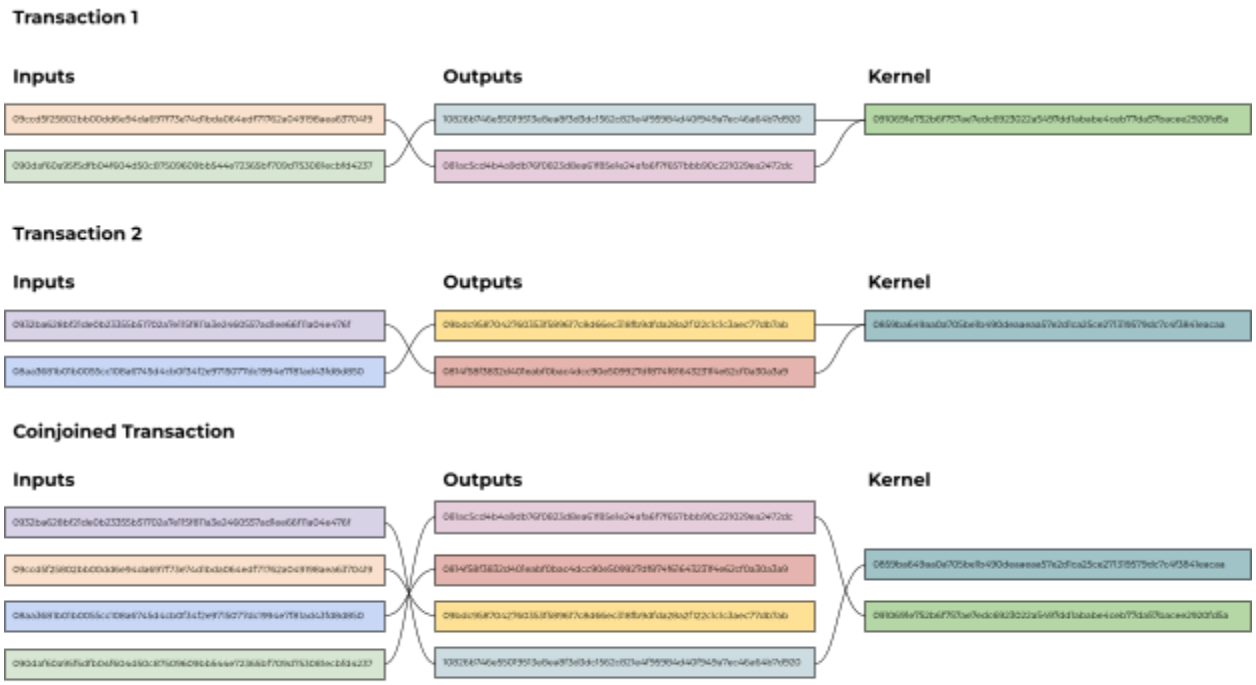
In the case of MWC, this allows us to verify that the sender has ownership of the coins they are sending and that no new coins are created in any transaction; this is all possible without ever knowing the amount in the transaction. So MWC retains its provable scarcity guarantees while providing additional privacy benefits.

Coinjoins

MWC also uses Coinjoins, or transaction aggregation. Coinjoins originate from bitcoin and allow for concatenating two transactions and turning them into a

single transaction by combining and mixing all the inputs and outputs. The net results are the same, but an observer cannot tell which outputs corresponded to which inputs.

In bitcoin, this is done through multiple users coming together interactively in order to actively combine their transactions; in MWC, it's part of the protocol and occurs naturally for all transactions and all blocks. Also in bitcoin, it's possible to identify coinjoin patterns because the amounts are public; in MWC, it's much more difficult to infer any such behavior.

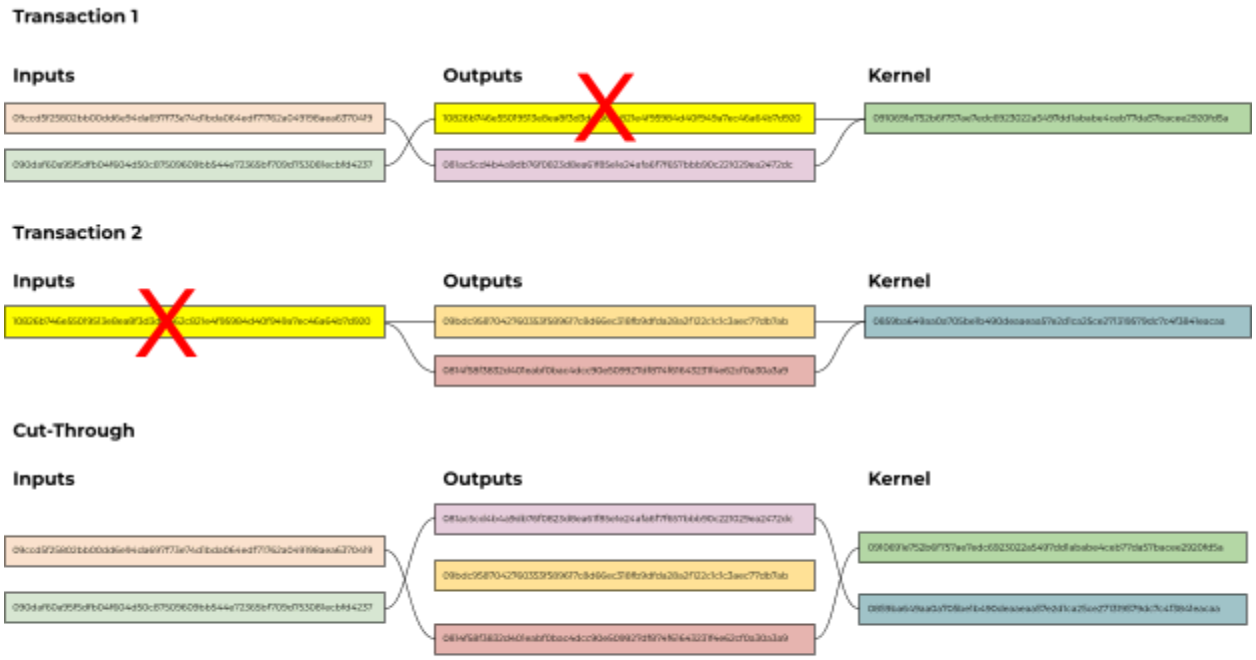


Coinjoins across two MWC transactions; the inputs, outputs and kernels are combined and the result is just one large transaction.

Coinjoins can be across any number of transactions, combining a myriad of inputs, output and kernels.

Cut-Throughs

On top of Confidential Transactions and Coinjoins, MWC uses Cut-Throughs. Once transactions are concatenated, intermediate inputs/outputs are removed, and the only remaining pieces of data are an initial set of inputs and a final set of outputs. When all is said and done, it's as if the intermediate outputs never existed.



Cut-Throughs across two MWC transactions; the intermediate input/output in yellow is removed and the result is just one transaction.

Cut-Throughs can be across any number of transactions, removing any number of intermediate inputs/outputs.

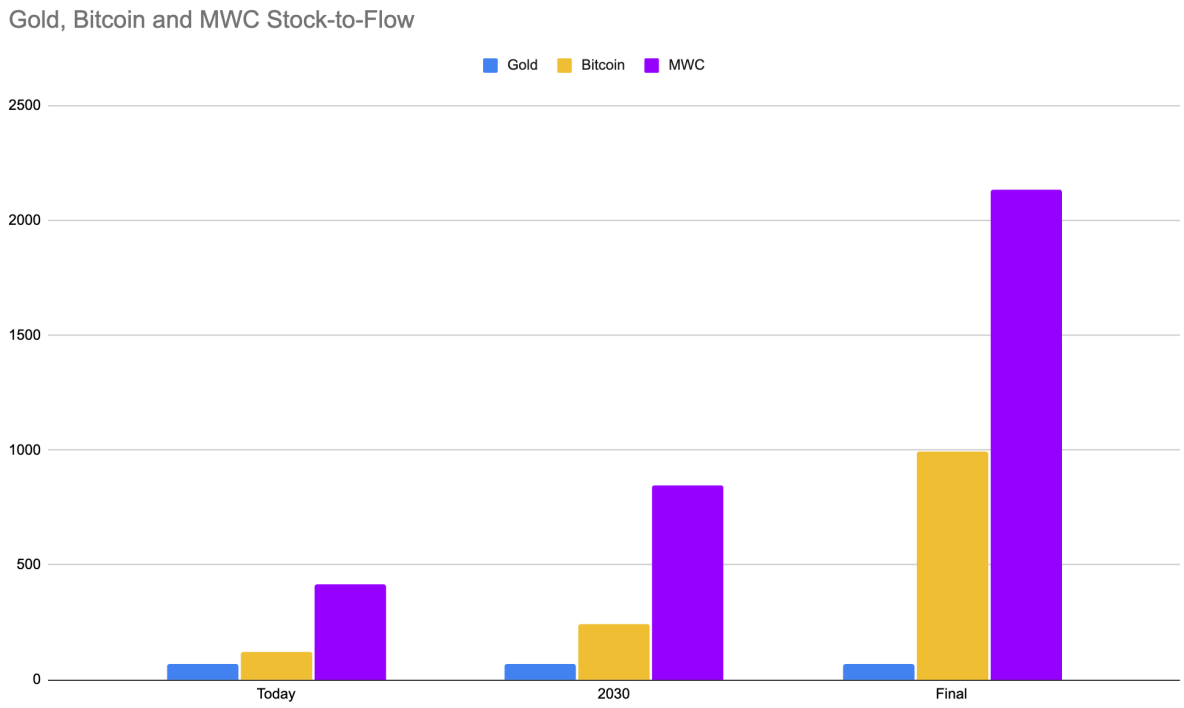
In addition to providing a high degree of privacy, Cut-Throughs enable unprecedented blockchain scalability. The MWC Initial Block Download (IBD) takes only 15-30 minutes on a standard Macbook laptop, which is more than an order of magnitude faster than Bitcoin.

This scalability enables a higher degree of monetary sovereignty, as individuals can run full-nodes much more easily and for much lower cost. This increases decentralization of the network and allows the tools built upon it to be built with a higher degree of individual ownership and freedom.

Limited Supply

Bitcoin became known as “digital gold” due in part to its fixed supply with a hard cap of 21 million bitcoin. MWC similarly has fixed supply, and is even more scarce than bitcoin with a maximum supply of 20 million and an emission rate of 0.05 MWC per block. This equates to a stock-to-flow ratio of 417 as compared to 120.

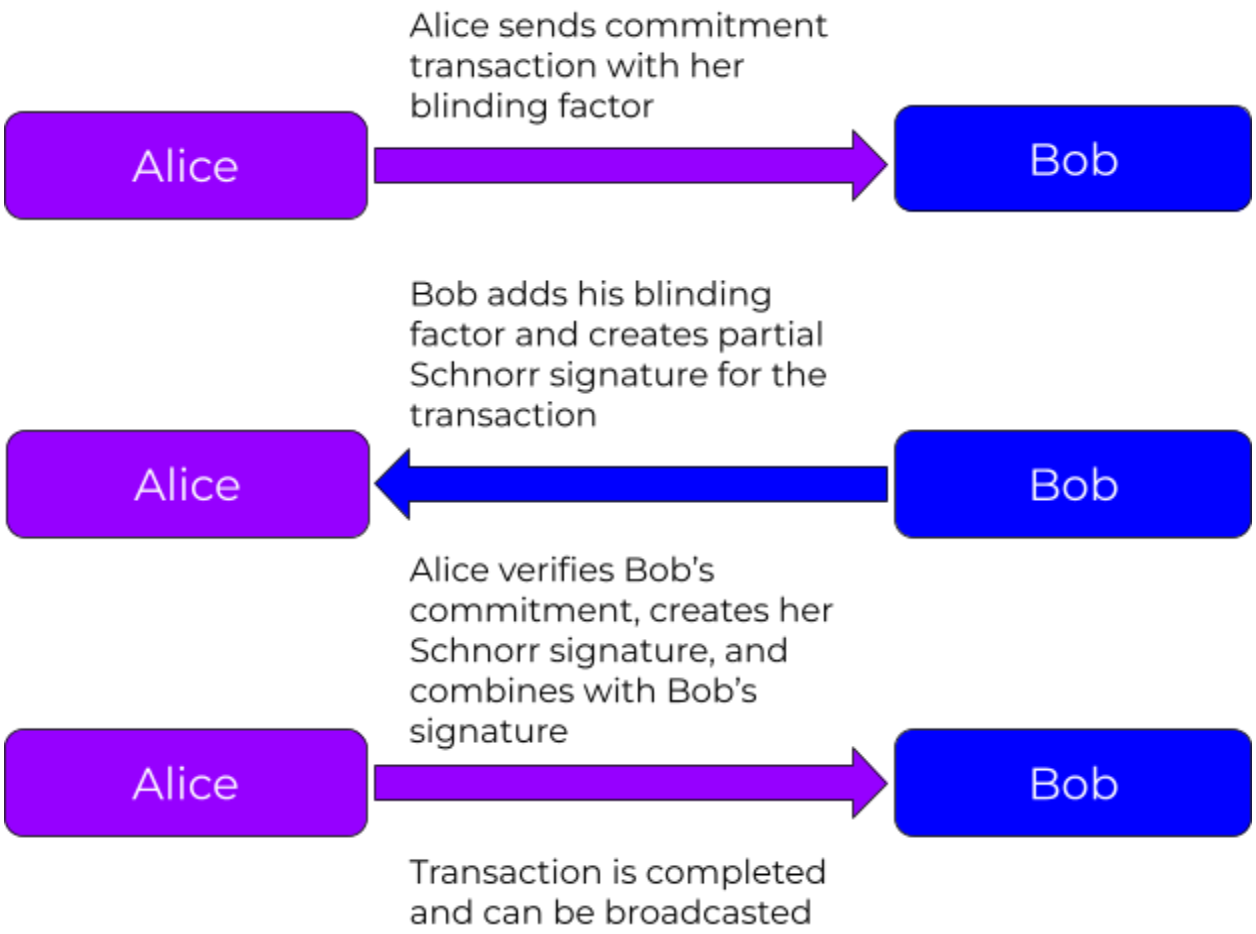
After the next round of emission rate decreases for bitcoin and MWC in 2028 and 2030, respectively, MWC stock-to-flow will be 845 and bitcoin's will be 240. In 2040, following the final MWC emission rate change, bitcoin's stock-to-flow will be 992 while MWC's will be 2135. MWC is, and will continue to be, the most scarce pure proof of work monetary good in the world.



Interactive Transactions

What are Interactive Transactions?

Interactive transactions are a method of transacting in which the sender signs a transaction and sends to the recipient; the recipient receives the transaction, adds their blinding factor, signs it and sends back; and then the sender signs the transaction again. Either side can then broadcast the final transaction.



The reason that transactions are interactive for Mimblewimble is because the amounts are obscured; the interactive nature is required in order to introduce the blinding factor which allows the amounts to be private while still providing the guarantees that the transaction is valid.

In practice, many transactions in the cryptocurrency space are interactive. For example, before someone sends a bitcoin transaction they will first send a test transaction and the recipient will confirm that the test was received. In the case of MWC, the interactivity is done cryptographically. As we'll see in the next section, there are benefits to interactive transactions.

Interactive Transactions and Monetary Sovereignty

In the case of bitcoin, any user can send bitcoin to any bitcoin address, without the consent of the recipient. This can result in tainted bitcoin, such as bitcoin from a hack, being sent to you and resulting in your address getting blacklisted.

It can also result in dusting attacks, where very tiny amounts of bitcoin are sent to your address. In order to consolidate and spend those amounts, you risk exposing links between the address with your dust and your other addresses, resulting in a leak of privacy.

Interactive transactions increase the surface of monetary sovereignty because they guarantee that you only receive MWC to an account or wallet that you control and when you have signed to receive it. This means that no user can send you MWC without your consent.

Running a Node

Running an MWC node is very easy due to the lightweight nature of the MWC blockchain. Because of cut-throughs and other optimizations, the MWC blockchain can be synced on commodity hardware within 15-30 minutes. In order to sync a full-node, simply download the latest release.

Download the version for whichever hardware and OS you're running the node on, and then proceed to run the software. It will connect to peers on the network for the initial block download, which may take up to 15-30 minutes, and then the node will remain synced, relaying transactions on the network.

The node can be used as the embedded node for running the desktop wallet application, or it can simply be used to fortify the network and contribute to the decentralization. In all cases, running a node is a step towards flexing your monetary sovereignty, and the scalability of MWC enables anyone to run a node more easily and for lower cost.

Mining

MWC is a pure proof-of-work blockchain which means that transactions are validated and included in a block by miners who are using energy and computer hardware to solve proof-of-work problems. MWC utilizes a proof-or-work algorithm called Cuckoo; in particular, MWC uses C31, which means 2^31 edges for the Cuckoo Cycle.

The choice of Cuckoo is by design, as the C31 algorithm relies more on memory than on computation, meaning that it's more difficult for hardware-specific solutions to gain a stronghold over the hashpower because ASICs won't be able to disproportionately solve the cycle faster.

The most common equipment for mining is G1s and G1 minis, manufactured by iPollo. However, there are GPU miners as well and eventually it is anticipated that people will be able to mine on commodity hardware such as the latest Apple Macbook Pro or desktop computer.

There are a number of mining pools that one can connect to in order to decrease the variance of their mining payout. By participating as part of a mining pool, miners can expect to receive payouts more frequently than if they were attempting to mine solo and win a block themselves; the payout is proportional to the hash rate contributed to the pool.

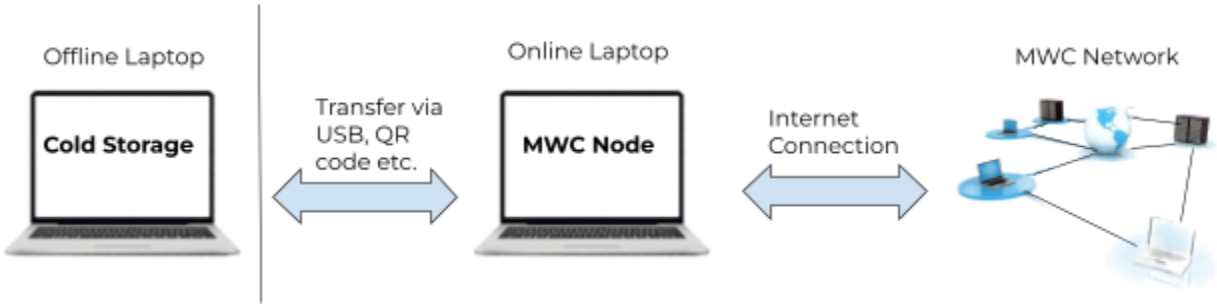
Self-Custody

For all the benefits that MWC provides as it relates to monetary sovereignty, one can only truly flex their monetary sovereignty muscle if they run their own full-node and self-custody their own private keys. Not your node; not your rules. Not your keys; not your coins.

The MWC desktop wallet, known as the QT wallet, provides an easy way to hold funds using commodity hardware. Simply download the latest version of the QT

wallet and follow the prompts to set up your wallet. Make sure to safely backup your seed phrase.

The QT wallet also enables proper cold storage. Cold storage protects funds by not exposing them to the threats of online attackers. In order to set up a cold storage wallet, follow the instructions here.



Finally, for users who prefer to use a hardware wallet like Ledger, there is MWC support for that as well. See here for how to use the MWC web wallet along with Ledger in order to hold your funds on your Ledger device.

Send/Receive

Transaction Construction

MWC Transactions contain a set of inputs, a set of outputs, and a transaction kernel. Inputs and outputs are similar in nature to bitcoin, except that they do not contain any meaningful public information. The transaction kernel is unique to Mimblewimble; the kernel is the key to providing the transaction guarantees. It serves as proof that the transaction is valid by providing two main things:

1. It proves that no new coins were created as part of this transaction, guaranteeing the fixed supply of MWC.
2. It provides the assurance that the transaction is authentic, meaning that it contains valid signatures from the input and output owners.

An MWC transaction is powerful in the information it contains and the value that can be transferred, all while leaving no discernable trail on the public blockchain except for the obfuscated outputs and the guarantees provided by the kernel.

Encrypted Slatepacks

MWC Transactions are slightly different from bitcoin due to the fact that transactions are interactive and there are no addresses. For many, a superior way to transact is through Encrypted Slatepack. In this section, we'll describe how Encrypted Slatepacks work.

Slatepack is a transaction standard that contains all the components for a Mimblewimble transaction and enables cryptographically secure interactive transactions.

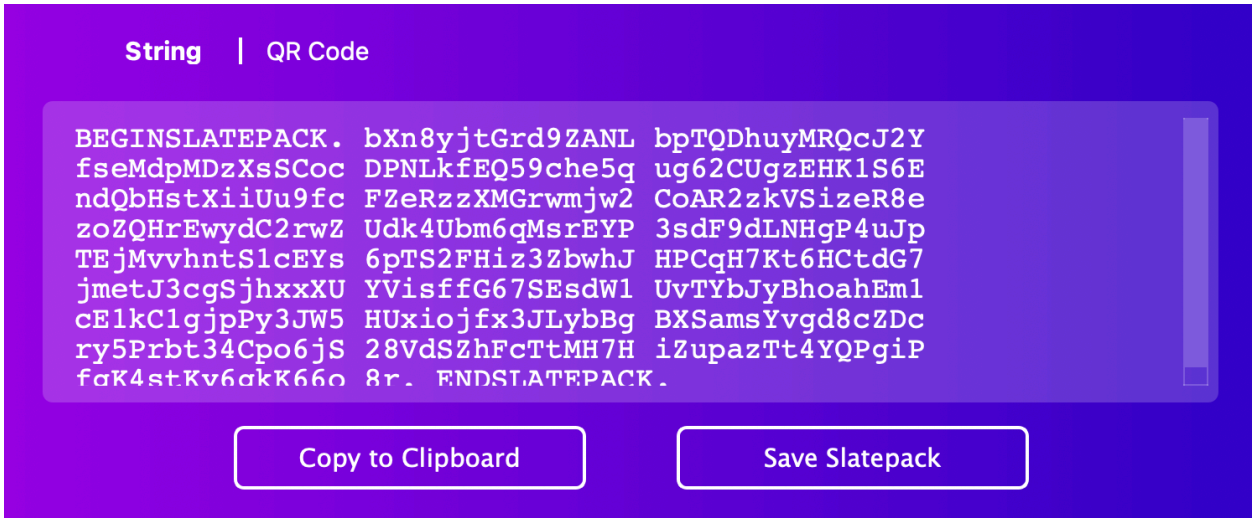
In order to send a Slatepack, a user first needs to generate a Slatepack in their wallet. This can be done through the 'Send' section of the QT Wallet. While the Slatepack can be unencrypted, best practice is to encrypt the Slatepack with the recipient's public Slatepack address.

An example encrypted Slatepack looks like this:

```
BEGINSLATEPACK. 3BFQfSV5ioaiMiN 3CyzZxcRpiQdKnZ mqwB393yvJZZPX6 JgrEjNDgfJ2kDf4
rQkR2NShWYKLwt9 ABRW2UAEhfXkLrh dFKv159JJ4mmcmz Le2Xf5CnL3o6hXj Yjxtk4cU8nPBDuf
PNKHUWmf8iEVLD5 z73YhieLVq6F277 tMChY6CVKfiVFGZ 4dm84i6NiXYrq1Q Ckh83WgGjywxrSV
7RKtSYSjGyQ3ENs 4S7Q9y4q5PtFvx8 RybYMczcv1JMbk4 tp7hyJooSa8hTgo YNBtrkA2xMMktpY
g3SRYddHicQK6aE e6gf11QjLbc9ao6 3sizD8BqZoxYLDr j7SWtdFzXqHCz6u Cfgb5V6Uzg58xNK wx4H.
ENDSLATEPACK.
```

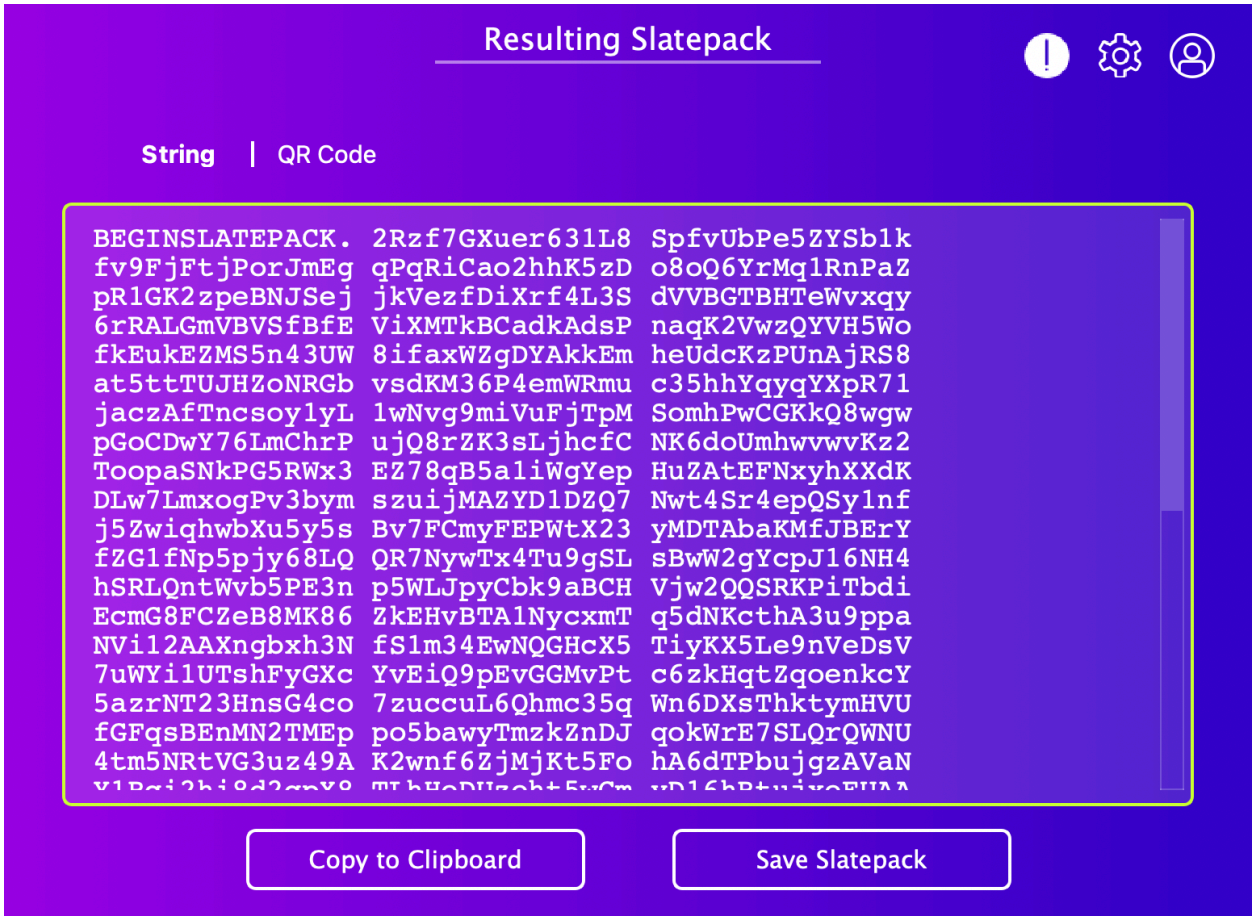
The steps to complete a MWC transaction using Encrypted Slatepack are:

1. The sender copies the Slatepack and sends it out-of-band to the recipient such as via email, via a USB drive, etc.

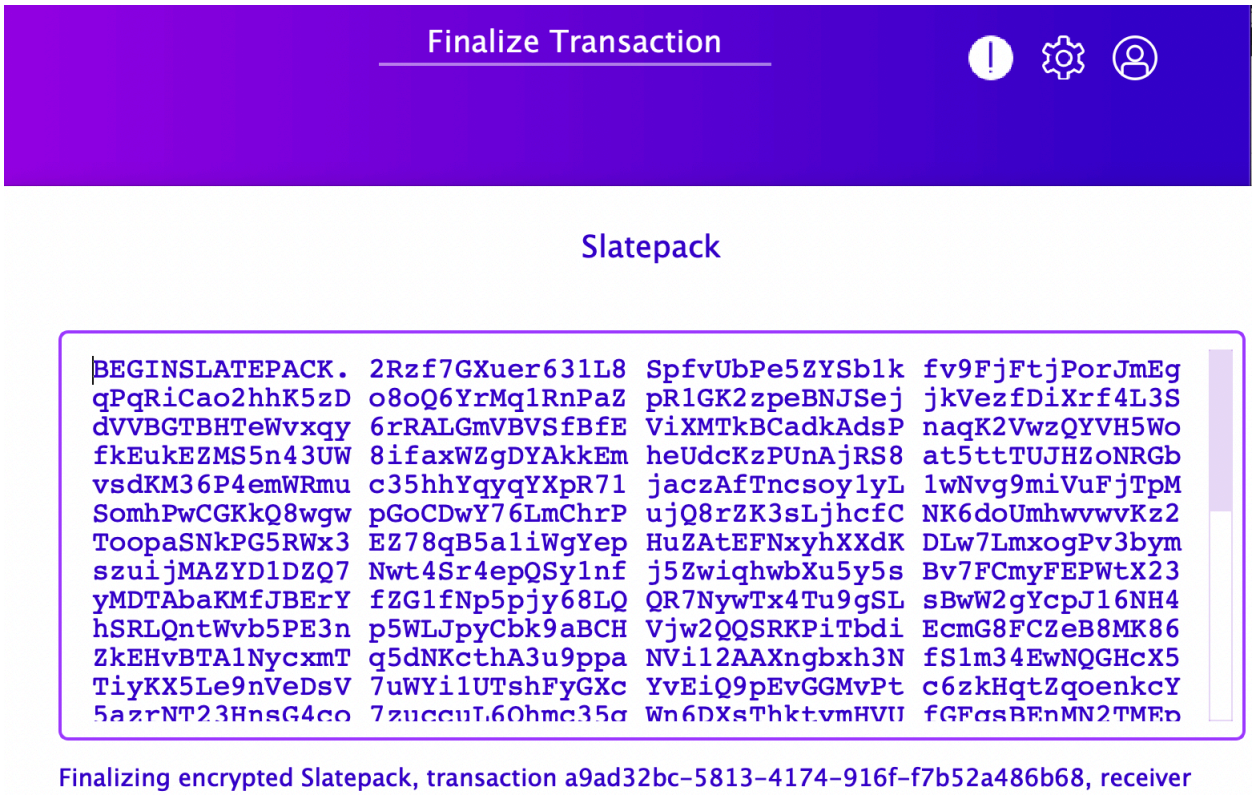


2. The recipient receives the encrypted Slatepack, pastes it in their wallet in the ‘Receive’ section of the QT wallet and generates a Slatepack response which contains their signature.

3. The recipient copies this Slatepack response and sends it back to the Sender out-of-band.


Resulting Slatepack
String | QR Code
BEGINSLATEPACK. 2Rzf7GXuer631L8 SpfvUbPe5ZYSb1k
fv9FjFtjPorJmEg qPqRiCao2hhK5zD o8oQ6YrMq1RnPaZ
pR1GK2zpeBNJSej jkVezfDiXrf4L3S dVVBGTBHTeWvxqy
6rRALGmVBVSfBfE ViXMTkBCadkAdsP naqK2VwzQYVH5Wo
fkEukEZMS5n43UW 8ifaxWZgDYAkkEm heUdcKzPUnAjRS8
at5ttTUJHZoNRGb vsdKM36P4emWRmu c35hhYqyqYXpR71
jaczAfTncsoy1yL 1wNvg9miVuFjTpM SomhPwCGKkQ8wgw
pGoCDwY76LmChrP ujQ8rZK3sLjhcfC NK6doUmhwvwvKz2
ToopaSNkPG5RWx3 EZ78qB5a1iWgYep HuZAtEFNxyhXXdK
DLw7LmxogPv3bym szuijMAZYD1DZQ7 Nwt4Sr4epQSy1nf
j5ZwiqhwbXu5y5s Bv7FCmyFEPWtX23 yMDTAbaKMfJBErY
fZG1fNp5pjy68LQ QR7NywTx4Tu9gSL sBwW2gYcpJ16NH4
hSRLQntWvb5PE3n p5WLJpyCbk9aBCH Vjw2QQSRKPiTbdi
EcmG8FCZeB8MK86 ZkEHvBTA1NycxmT q5dNKcthA3u9ppa
NVi12AAXngbxh3N fS1m34EwNQGHcX5 TiyKX5Le9nVeDsV
7uWYi1UTshFyGXc YvEiQ9pEvGGMvPt c6zkHqtZqoenkcY
5azrNT23HnsG4co 7zuccuL6Qhmc35q Wn6DXsThktymHVU
fGFqsBEnMN2TMEp po5bawyTmzkZnDJ qokWrE7SLQrQWNU
4tm5NRtVG3uz49A K2wnf6ZjMjKt5Fo hA6dTPbujgzAVaN
Copy to Clipboard
Save Slatepack

4. The Sender then copies the final Slatepack into their wallet and finalizes the transaction.



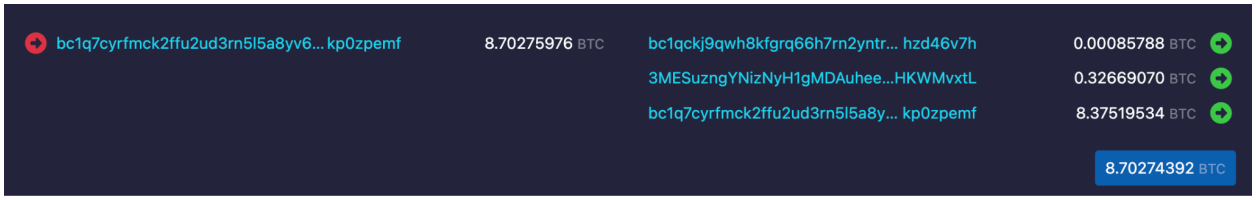
Finalizing encrypted Slatepack, transaction a9ad32bc-5813-4174-916f-f7b52a486b68, receiver

The Slatepack contains all the transaction components and signatures, and the encrypted nature of it ensures that it's not subject to a man-in-the-middle attack.

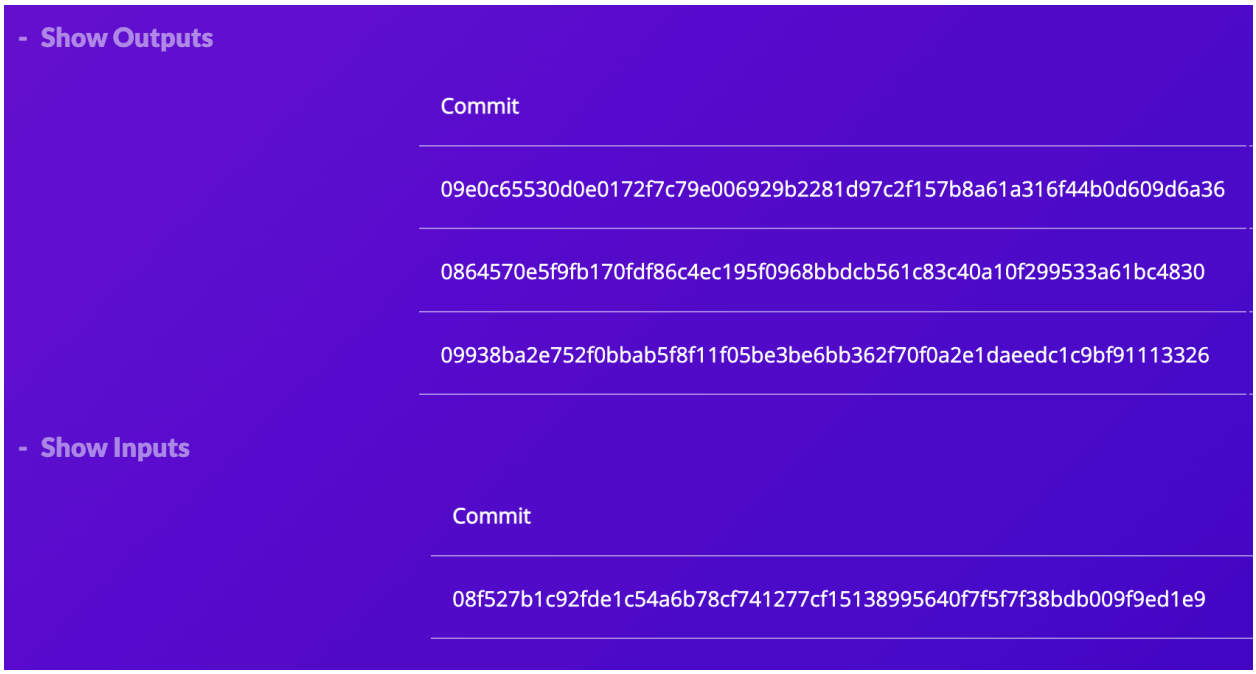
Payment Proofs

All transaction details on the bitcoin blockchain are public, meaning that one can point to a public transaction to verify that a certain amount was sent from a certain address as part of a transaction.

Bitcoin Transaction Showing Amounts and Addresses:



MWC Transaction with No Addresses or Amounts Revealed:



When addresses don't exist and amounts are private, how can we provide these same assurances?

The answer is that MWC has payment proofs. Payment proofs prove that a certain amount of funds were provided as part of a transaction. The proof is generated by the sender and can be provided as proof that payment was sent. Additionally, a message can be included in the transaction, so, this could be applied in a way that would enable financial institutions to easily comply with the Travel Rule.

The sender can generate the proof from the Transactions section of their QT wallet; the receiver can then verify the proof by uploading it to the QT wallet or by uploading it to a block explorer such as https://explorer.mwc.mw/.

Obtaining MWC

Exchanges

As with bitcoin, the most common way to obtain MWC is by purchasing on an exchange. The most commonly traded pairs are MWC/BTC and MWC/USDT, meaning that one is exchanging BTC or USDT for MWC.

Exchanges that support MWC include Whitebit, XT, and Ascendex.

Peer-to-Peer Transactions

As private money, MWC is conducive to direct, bilateral transactions. Two people can exchange encrypted Slatepacks in order to discreetly transfer funds between each other.

Over time, it is anticipated that there will be increased mobile wallet support, including QR code support and the ability to run a full node on mobile. This will enable further support for these types of transactions and MWC usage.

Imagine: a pure proof of work private and fungible ossified cryptocurrency with 417 stock-to-flow with users running a full-node on their mobile device.

Atomic Swaps

Because of the ease of running a full node, MWC also lends itself well to atomic swaps. Users with a bitcoin node and an MWC node can easily generate, sign and settle atomic swaps without the need for any intermediary or 3rd party.

Atomic swaps are a powerful way to exchange BTC for MWC, especially in a world where these two complementary tools for monetary sovereignty continue to gain market share and dominance.

Auditing MWC Supply

There is a belief among blockchain participants that there is a tradeoff between privacy and auditability. At first glance, this tradeoff makes sense as a blockchain that obscures amounts would be notably difficult to audit. In particular, it would be difficult to prove that no new coins were created in a particular block or over the lifetime of the chain.

MWC uses Pederson Commitments which enable unique, elegant proofs where one can prove that every block is valid and that no new coins have been created.

Per-Block Verification

To verify each block, one can simply compute the equation:

Σinputs - Σoutputs == kernel excesseses + overage

The kernel is proof that there were valid signatures for the transaction and that the sum of blinding factors is equal to the blinding factors used across all the inputs. So, one can verify the integrity of the amounts while the amounts themselves remain blinded.

This can be run on any single block using an archival node. The equation is computed in commitment space and the result is a verification per block that no new coins were created. A sample result looks like:

```
08ad5d8c3f...: Output ✅
🔍 Requesting block at height 2886795
overage: 50000000
kernel offset: 1d452cb1e0b76054b755f15f1325e9a1df2ad99d3f08ef05b2955916eb3411a0
input hexes: ['097c1f4ef404f52a853e878084584b8250c72093bda9d54ac9b14afe45200550b2', '0946fa9b5a5975e33e6273eb0a93eadbec67b245849e589ff94eb92f19360a6b3a']
output hexes: ['09e0dc141c9e29f78cfe9a4b556f1f25a844a3f61c984d9139b9f763fae9f7f70e', '09f409e055b643505796d2315e30743095a43463a7c1fd7b92dc2c9d3a25f64e01']
kernel_offset_block_minus_119c3dbcd540fd2fc3f27e5027477faa966d80f66daf5075580c473f359c3d2ff
rhs partial: PublicKey(de36555bdbd7c1916fcf5a96ddf16bbebbaa721c72b23c174d145e07ebdef032de3a76bef919f442656f5ca48b72e36d9df87fd9d8365b5188c74e0119e2db0d)
outputs - inputs: PublicKey(5faeec37cf17b96433f15fccf56d7ea289ef5911076d927f8ee0e09e90dd3bf70677ddf422e0d8a8087243b25fbdf7757bb6be07b0ec2d83a5ae9be87a7cf352)
kernel excesses + overage: PublicKey(5faeec37cf17b96433f15fccf56d7ea289ef5911076d927f8ee0e09e90dd3bf70677ddf422e0d8a8087243b25fbdf7757bb6be07b0ec2d83a5ae9be87a7cf352)
Equal? true
✅ Block verified! Block 2886795
```

For full code, see Appendix A.

Full Blockchain Verification

With MWC, we can take the verification a step further. The offset incorporates the sum of offsets across all blocks. So we can compute the equation:

ΣUTXO commitments == Σkernel excesses + total_offset * G + total_reward * H

And verify that all the UTXOs are equivalent to the sum of excesses and rewards across all blocks! This computation is much faster than going block-by-block and comes with the same cryptographic guarantees. A sample result looks like:

```
Collected 3109884 kernel excesses
Total UTXOs collected: 72064
Total MWC block reward as of height 2944628: 10984886.967788020 MWC
ΣExcesses: 081483f0f1610c1f85635d13716d155ac2508ae2be3370d766b95d03a2d9c9c863
Offset   : 09e5db49ab09be87cc4668ec3b0557ded81f15a75637b3ba29bf53792dfe35885e
Reward   : 08a08c1d903991b65a7942d79453950c314214985c0c5ce01a2329e5bc2beb2bb0
LHS (ΣUTXO commitments): 088be858ca70a7359f0d660db8d23933399c9fe5d0547e62d0f6ff2204e04c3efc
RHS (Σexcess + offset + reward): 088be858ca70a7359f0d660db8d23933399c9fe5d0547e62d0f6ff2204e04c3efc
✅ MWC supply balances!
```

For full code, see Appendix B.

Conclusion

This guide provides an overview of MWC and its potential. MWC uniquely enables things that were not possible before due to its privacy, scarcity and scalability.

Users can flex their monetary sovereignty through the ease of running a full node, and they can transact privately on a decentralized, public and pure proof-of-work blockchain while holding their money in the highest stock-to-flow asset on the planet.

This is the power extremely scarce ghost money enables for its users.

Appendix A

Per-block verifier

#Per block supply verifier Σinputs - Σoutputs == kernel excesseses + overage

```
import requests
from requests.auth import HTTPBasicAuth
import json
from dotenv import load_dotenv
import os
from mwc_commit import verify_block_kernel_sum, scalar_subtract_hex, verify_output_range_proof

load_dotenv()

OWNER_SECRET = os.getenv('OWNER_SECRET')

foreign_api_url = "http://127.0.0.1:3413/v2/foreign"
foreign_api_user = 'mwcmain'
foreign_api_password = OWNER_SECRET

def get_overage_nano(height: int) -> int:
    """
    Returns the overage in nanocoins for a given block height.
    Overage = block_reward * 1_000_000_000
    """
    reward_schedule = [
        (0,         2.380952380),
        (212_580,   0.600),
        (385_380,   0.450),
        (471_780,   0.300),
        (644_580,   0.250),
        (903_780,   0.200),
        (1_162_980, 0.150),
        (1_687_140, 0.100),
        (2_211_300, 0.050),
        (5_356_260, 0.025),
```

```
        (10_597_860, 0.010),
        (886_947_008, 0.0),  # Final cap
    ]

    for i in range(len(reward_schedule) - 1, -1, -1):
        if height >= reward_schedule[i][0]:
            reward = reward_schedule[i][1]
            return int(reward * 1_000_000_000)

    # Should not reach here unless height is negative
    return 0

# Initialize counters outside your block loop
coinbase_only_count = 0
verified_count = 0
failed_count = 0

def get_block_and_verify(height, api_url, username, password):
    global coinbase_only_count, verified_count, failed_count
    payload = {
        "jsonrpc": "2.0",
        "method": "get_block",
        "params": [height, "", ""],
        "id": 1
    }

    print(f"Requesting block at height {height}")

    response = requests.post(foreign_api_url, json=payload, auth=HTTPBasicAuth(foreign_api_user, foreign_api_password))

    result = extract_block_data(response.text)

    input_hexes = result["input_hexes"]
    output_hexes = result["output_hexes"]
    kernel_hexes = result["kernel_hexes"]
    kernel_offset = result["kernel_offset"]
    output_commit_proof_pairs = result["output_commit_proof_pairs"]
    overage = get_overage_nano(height)
```

```
    print("overage: " + str(overage))

    payload_block_minus_1 = {
        "jsonrpc": "2.0",
        "method": "get_block",
        "params": [height - 1, "", ""],
        "id": 1
    }
    response_block_minus_1 = requests.post(foreign_api_url, json=payload_block_minus_1,
auth=HTTPBasicAuth(foreign_api_user, foreign_api_password))
    result_block_minus_1 = extract_block_data(response_block_minus_1.text)
    kernel_offset_block_minus_1 = re
sult_block_minus_1["kernel_offset"]

    print("kernel offset: " + str(kernel_offset))
    print("input hexes: " + str(input_hexes))
    print("output hexes: " + str(output_hexes))
    print("kernel_offset_block_minus_1" + str(kernel_offset_block_minus_1))

    # Inside your loop over block heights
    if kernel_offset == kernel_offset_block_minus_1:
        print("Coinbase-only block, no kernel sum verification needed.")
        coinbase_only_count += 1
    else:
        new_offset = scalar_subtract_hex(kernel_offset, kernel_offset_block_minus_1)
        try:
            result = verify_block_kernel_sum(
                input_hexes=input_hexes,
                output_hexes=output_hexes,
                kernel_hexes=kernel_hexes,
                offset_hex=new_offset,
                overage=overage,
            )
            if result:
                print(f"✅ Block verified! Block {height}")
                verified_count += 1
            else:
                print(f"❌ Verification failed. {len(input_hexes)} inputs in block {height}")
```

```
            failed_count += 1
        except Exception as e:
            print(f"❌ Verification failed at block height {height}: {e}")
            failed_count += 1

    # Print running summary after each block
    print(f"📊 Summary so far → Coinbase-only: {coinbase_only_count}, Verified: {verified_count}, Failed: {failed_count}\n")
    for commit, proof in output_commit_proof_pairs:
        result = verify_output_range_proof(commit, proof)
        print(f"{commit[:10]}...: {'Output ✅' if result else '❌'}")

def extract_block_data(raw_json_text):
    # If it's a raw string (e.g., from repr or logs), clean and load it
    if isinstance(raw_json_text, str):
        try:
            # Handle escape sequences like \n, \"
            block_json = json.loads(raw_json_text)
        except json.JSONDecodeError:
            block_json = json.loads(bytes(raw_json_text, "utf-8").decode("unicode_escape"))
    else:
        block_json = raw_json_text

    block_data = block_json["result"]["Ok"]

    # Extract kernel offset
    kernel_offset = block_data["header"]["total_kernel_offset"]

    # Extract inputs
    input_hexes = block_data.get("inputs", [])

    # Extract outputs: list of dicts with commit and proof
    outputs = block_data.get("outputs", [])
    output_hexes = [o["commit"] for o in outputs]
    output_commit_proof_pairs = [(o["commit"], o["proof"]) for o in outputs]

    # Extract kernel excesses
    kernels = block_data.get("kernels", [])
    kernel_hexes = [k["excess"] for k in kernels]
```

```
    return {
        "input_hexes": input_hexes,
        "output_hexes": output_hexes,
        "kernel_hexes": kernel_hexes,
        "kernel_offset": kernel_offset,
        "output_commit_proof_pairs": output_commit_proof_pairs,
    }

if __name__ == '__main__':

    # height_to_check = 2211348 #to test input_commitments error
    # get_block_and_verify(height_to_check, foreign_api_url, foreign_api_user, foreign_api_password)

    start_height = 2928295
    end_height = 2928480

    for height_to_check in range(start_height, end_height):
        get_block_and_verify(height_to_check, foreign_api_url, foreign_api_user, foreign_api_password)
```

Appendix B

```
//Rust supply verifier ΣUTXO commitments == Σkernel excesses + total_offset * G + total_reward * H
use mwc_core::core::{OutputIdentifier, TxKernel, block::BlockHeader};
use mwc_core::core::pmmr::{self, ReadonlyPMMR, ReadablePMMR};
use mwc_core::ser::ProtocolVersion;
use mwc_core::global::{self, ChainTypes};
use mwc_store::pmmr::PMMRBackend;
use mwc_chain::store::ChainStore;
use secp256k1zkp::{Secp256k1, ContextFlag};
use secp256k1zkp::pedersen::{Commitment, Commitment as ZkpCommitment};
use secp256k1zkp::key::SecretKey as ZkpSecretKey;
use mwc_core::core::hash::{Hash, Hashed};
use std::path::Path;
use std::error::Error;
use std::sync::Arc;

use mwc_chain::txhashset::TxHashSet;
use mwc_core::consensus::{calc_mwc_block_overage, MWC_BASE};
pub fn get_total_block_reward_as_of_height(target_height: u64) -> u64 {
    calc_mwc_block_overage(target_height, true)
}

fn main() -> Result<(), Box<dyn Error>> {
    global::set_local_chain_type(ChainTypes::Mainnet);
    let secp = Secp256k1::with_caps(ContextFlag::Commit);

    let chain_path = Path::new("/Users/juthicamallela/.mwc/main/chain_data");
    let store = Arc::new(ChainStore::new(
        Path::new("/Users/juthicamallela/.mwc/main/chain_data").to_str().unwrap(),
    )?);
    let mut header = store.head_header()?;
    let tip_height = header.height;

    println!("🔍 Tip height: {}", tip_height);
```

```
// Open TxHashSet
let txhashset_path = "/Users/juthicamallela/.mwc/main/chain_data".to_string();
let txhashset = TxHashSet::open(txhashset_path, store.clone(), None, &secp)?;

// Output PMMR
let output_pmmr = txhashset.output_pmmr_at(&header);

let mut utxo_commits: Vec<ZkpCommitment> = vec![];
let mut count = 0;

for pos in output_pmmr.leaf_pos_iter() {
    if let Some(output_id) = output_pmmr.get_data(pos) {
        let commit = output_id.commitment();

        // Only include unspent outputs (true UTXOs)
        if let Ok(Some((_out, commit_pos))) = txhashset.get_unspent(commit) {
            println!(
                "Block {:<6} | Commit: {}",
                commit_pos.height,
                hex::encode(commit.0)
            );
            utxo_commits.push(ZkpCommitment(commit.0));
            count += 1;
        }
    }
}

println!("Total unspent outputs collected: {}", count);

let lhs_commit = secp.commit_sum(utxo_commits, vec![])?;

// ---- RHS: Kernel Excess Commitments + Offset + Reward ----
let mut kernel_excess_commits: Vec<Commitment> = Vec::new();
let mut actual_collected_kernels = 0;

loop {
    let block = store.get_block(&header.hash())?;
```

```
    for kernel in block.kernels() {
        let commit = ZkpCommitment(kernel.excess.0);
        kernel_excess_commits.push(commit);
        actual_collected_kernels += 1;
    }

    if header.height % 100000 == 0 {
        println!("Processing excesses: {}", header.height);
    }

    if header.height == 0 {
        break;
    }

    header = store.get_previous_header(&header)?;
}

println!("Collected {} kernel excesses ", actual_collected_kernels);
let sum_excesses_commit = secp.commit_sum(kernel_excess_commits, vec![])?;

// Kernel offset
let offset_bf = store.head_header()?.total_kernel_offset;
let offset_sk = ZkpSecretKey::from_slice(&secp, offset_bf.as_ref())?;
let offset_commit = secp.commit(0, offset_sk)?;

let total_reward = get_total_block_reward_as_of_height(tip_height);

println!("Total UTXOs collected: {}", count);
println!("Total MWC block reward as of height {}: {}.{:09} MWC",
        tip_height,
        total_reward / MWC_BASE, // MWC_BASE is also defined in consensus
        total_reward % MWC_BASE);

let reward_commit = secp.commit_value(total_reward)?;

let rhs_commit = secp.commit_sum(vec![sum_excesses_commit, offset_commit, reward_commit], vec![])?;

println!("ΣExcesses: {}", hex::encode(sum_excesses_commit.0));
```

```
    println!("Offset   : {}", hex::encode(offset_commit.0));
    println!("Reward   : {}", hex::encode(reward_commit.0));

    println!("LHS (ΣUTXO commitments): {}", hex::encode(lhs_commit.0));
    println!("RHS (Σexcess + offset + reward): {}", hex::encode(rhs_commit.0));

    if lhs_commit == rhs_commit {
        println!("✅ MWC supply balances!");
    } else {
        println!("❌ MWC supply mismatch!");
    }

    Ok(())
}
```