# Mimblewimble

Andrew Poelstra*

2016-10-06 (commit `e9f45ec`)

**Abstract**

At about 04:30 UTC on the morning of August 2nd, 2016, an anonymous person using the name Tom Elvis Jedusor signed onto a Bitcoin research IRC channel, dropped a document hosted on a Tor hidden service[Jed16], then signed out. The document, titled Mimblewimble, described a blockchain with a radically different approach to transaction construction from Bitcoin, supporting noninteractive merging and cut-through of transactions, confidential transactions, and full verification of the current chainstate without requiring new users to verify the full history of any coins.

Unfortunately, while the paper was detailed enough to comminicate its main idea, it contained no arguments for security, and even one mistake[1]. The purpose of this paper is to make precise the original idea, and add further scaling improvements developed by the author.

In particular, Mimblewimble shrinks the transaction history such that a chain with Bitcoin's history would need 15Gb of data to record every transaction (not including the UTXO set, which including rangeproofs, would take over 100Gb). Jedusor left open a problem of how to reduce this; we solve this, and combine it with existing research for compressing proof-of-work blockchains, to reduce the 15Gb to less than a megabyte.



*`grindelwald@wpsoftware.net`

[1] `https://www.reddit.com/r/Bitcoin/comments/4vub3y/mimblewimble_noninteractive_coinjoin_and_better/d61n7yd`

# Contents



# 1 Introduction

In 2009, Satoshi Nakamoto introduced the Bitcoin cryptocurrency[Nak09], an online currency system to allow peer-to-peer transfer of digital tokens. These tokens' ownership is ascertained by the use of public keys, and transfer is accomplished by means of a public ledger, a globally replicated list of transactions which destroy *unspent transaction outputs* (UTXOs) and create new ones of equal value but different keys.

All Bitcoin coins originate in *coinbase transactions*, transactions which create new outputs without destroying any old ones, and which are limited in value to maintain a fixed inflation schedule. Once created, they change hands by means of ordinary transactions. This means that new users, in order to fully verify that their view of the system is uncompromised (absent theft or illegal inflation), must download and validate the entire history from the system's genesis in 2009 until today. In other words, they must download and "replay" every transaction that has ever occurred.

Today the Bitcoin blockchain sits just shy of 100Gb on the author's disk. Had Bitcoin used Confidential Transactions[Max15] (CT), each of the approximately 430 million outputs would consume 2.5Kb each, totalling over a terabyte of historical data[2].

In this paper, we describe a research project to design a Bitcoin-like cryptocurrency, which achieves dramatically better scaling and privacy properties than Bitcoin. In particular, it allows

[2]The author is aware of unpublished research that would reduce this quantity by about 25%; the total is still formidable.

removal of most historic blockchain data, including spent transaction outputs, rangeproofs and all, while still allowing users to fully verify the chain. Indeed, it is possible for the system to function even if no users retain the vast majority of historic blockchain data.

Further, this currency allows transactions to be noninteractively combined (*a la* Coinjoin[Max13a]) and cut-through[Max13b], eliminating much of the topological structure of the transaction graph. If this is done without publishing the original component transactions, the result is an enormous boon to user privacy, amplified by CT.

Unfortunately, it accomplishes these goals at cost of sacrificing functionality.

## 1.1 Overview and Goals

Mimblewimble is a design for a cryptocurrency whose history can be compacted and quickly verified with trivial computing hardware even after many years of chain operation. As a secondary goal, it should support strong user privacy by means of confidential transactions and an obfuscated transaction graph. However, to achieve these goals, Mimblewimble cannot support a general-purpose scripting system such as that in Bitcoin. This precludes such functionality as zero-knowledge contingent payments[Max16], cross-chain atomic swaps[Nol13] and micropayment channels[PD16]. Further research is needed to emulate these functionalities on top of Mimblewimble, and is discussed in Section 4.

More precisely,

- Direct transfer of value between parti(es) to other parti(es) should be possible on the system.
- All transactions should use confidential transactions to blind their output amounts.
- Transactions should be noninteractively aggregable[Mou13], *i.e.* support a noninteractive variant of CoinJoin[Max13a, Max13b], such that parties not privy to the original transactions are unable to separate them. This can improve censorship resistance and privacy, though it is unclear how to design a safe peer-to-peer network capable of exploiting this ability.
- For a new participant, the amount of bandwidth and processing power needed to catch up with the system should be proportional to the *current state* of the system, *i.e.* the size of the UTXO set plus rangeproofs, rather than the total size of all historical transactions.

  As described in the next section, and more thoroughly in Section 3.3, Mimblewimble has a scheme for compressing blockchain history to a size polylog in the original size, which in practice for a system with Bitcoin's scale should allow a decade or more of transaction history to be verified in several seconds using a couple megabytes of data[3].

[3] Experiments with the compact chains described in this paper suggest that a 500000 block chain may have a compact length of around 300 blocks (though with high variance across multiple experiments), each containing sinking signatures which average ten 96-byte elliptic curve points, merkle commitments to previous blocks consisting of forty 40-byte (block-hash, difficulty) pairs, and some extra block header data, totalling less than 3Kb of data. Across 300 blocks this is 900Kb total of compact chain data. Verification time is dominated by pairing computations, of which there are on average ten per block, or 3000. On the author's laptop using Ben Lynn's `libpbc` library, these can be done in under 20 seconds using a single core. By more careful choice of elliptic curve, and focused optimization work, this number can surely be improved.

On the other hand, Bitcoin's current state of 40 million unspent outputs would grow to 100Gb and a few days of verification on a modern CPU, thanks to Mimblewimble's use of confidential transitions. It is an open problem how to improve this. We observe that even without cryptographic improvements, it would go a long way to simply cap the UTXO set size as a function of blockheight and let the transaction-fee market take care of it.

## 1.2 Trust Model

Like Bitcoin, Mimblewimble is a blockchain-based cryptocurrency intended to be instantiated in a decentralized manner in which users may join or leave the system at any time. Further, upon (re)joining the network, users should be able to determine the network state, at least up to some recent time, and verify that this state was obtained by a series of valid state transitions, without trusting the honesty of any third parties.

However, there are two points of departure from Bitcoin's trust model:

1. Unlike Bitcoin, whose blockchain describes every transaction in its entirety, and therefore allows all users to agree on and verify the precise series of transactions that led to the current chainstate, Mimblewimble only allows users to verify the essential features:

   - A transaction, once committed to the block, cannot be reversed without rewriting the block (or by the owner(s) of its outputs explicitly undoing it).
   - The current state of all coins was obtained by zero net theft or inflation: there are exactly as many coins in circulation as there should be, and there for each unspent output there exists a path of transactions leading to it, all of which are committed in the chain and authorized.

     Note that there may be other paths which have also been committed in the chain, during which some transactions may have been invalid or unauthorized or inflationary, but since a legitimate path exists, all these things must have netted out to zero.

2. Like Bitcoin, verifiers may see multiple differing blockchains, and select the valid one as the one with the greatest *total work*. This is detailed in Section 3.3.2.

   However, in Bitcoin the total work represents both the *expected work* to produce such a blockchain as well as the *proven work* of the chain, in the sense that any party who expends significantly less than this much work will be unable to produce such a chain except with negligible probability.

   Mimblewimble, on the other hand, separates these. The total work still represents the expected work to produce the blockchain, and therefore incentivizes rational actors to contribute to the most-work chain rather than rewriting it. The proven work, on the other hand, is capped at some fixed value independent of the length of the chain, and serves to make forgery by irrational lucky actors prohibitively expensive.

# 2 Preliminaries

## 2.1 Cryptographic Primitives

**Groups.** Throughout, $\mathscr{G}_1$, $\mathscr{G}_2$ will denote elliptic curve groups adorned with an efficiently computable bilinear pairing $\hat{e} : \mathscr{G}_1 \times \mathscr{G}_2 \to \mathscr{G}_T$, with $\mathscr{G}_T$ equal to the multiplicative group of $\mathbb{F}_{q^k}$ for some prime $q$, small positive integer $k$. We further require both $\mathscr{G}_1$ and $\mathbb{G}_2$ to have prime order $r$. Let $G, H$ be fixed generators of $\mathscr{G}_1$ whose discrete logarithms relative to each other are unknown (*i.e.* they are nothing-up-my-sleeve (NUMS) points; $\mathscr{G}_2$ does not need any canonical generators. We will make computational hardness assumptions about these groups as needed. We write $\mathbb{Z}_r$ for the integers modulo $r$, and write $+$ for the group operation in all groups.

All cryptographic schemes will have an implicit GenParams$(1^\lambda)$ phase which generates $r$, $\mathscr{G}_1$, $\mathscr{G}_2$, $\mathscr{G}_T$, $G$ and $H$ uniformly randomly given a security parameter $\lambda$.

### 2.1.1 Standard Primitives

**Definition 1.** *A* commitment scheme *is a pair of algorithms* Commit$(v,r) \to \mathscr{G}$, Open$(v,r,C) \to \{\text{true}, \text{false}\}$ *such that* Open$(v,r,\text{Commit}(v,r))$ *accepts for all* $(v,r)$ *in the domain of* Commit. *It must satisfy the following security properties.*

- Binding. *The scheme is* binding *if for all* $(v,r)$ *in the domain of* Commit*, there is no* $(v',r') \neq (v,r)$ *such that* Open$(v',r',\text{Commit}(v,r))$ *accepts. It is* computationally binding *if no PPT algorithm can produce such a* $(v',r')$ *with nonneglible probability.*
- Hiding. *The scheme is* (perfectly, computationally) hiding *if the distribution of* Commit$(v,r)$ *for uniformly random r is (equal, computationally indistinguishable) for different values of v.*

**Definition 2.** *We define a* homomorphic commitment scheme *as one for which there is a group operations on commitments and* Commit *is homomorphic in its value parameter. That is, one where commitments to v, v′ can be added to obtain a commitment to* $v+v'$ *having the same security properties.*

**Example 1.** *Define a* Pedersen commitment *as the following scheme:* Commit$: \mathbb{Z}_r^2 \to \mathscr{G}$ *maps* $(v,r)$ *to* $vH + rG$*, and* Open$: \mathbb{Z}_r^2 \times \mathscr{G} \to \{\text{true}, \text{false}\}$ *checks that* Commit$(v,r)$ *equals* $vH + rG$*.*

*If the discrete logarithm problem in* $\mathscr{G}$ *is hard, then this is a computationally binding, perfectly hiding homomorphic commitment scheme[Ped01].*

**Definition 3.** *Given a homomorphic encryption C, we define a* rangeproof *on C as a cryptographic proof that the committed value of C lies in some given range* $[a,b]$*. We further require that rangeproofs are zero-knowledge proofs of knowledge (zkPoK) of the opening information of the commitments.*

Unless otherwise stated, rangeproofs will commit to the range $[0, 2^n]$ where $n$ is small enough that no practical amount of commitments can be summed to produce an overflow.

We may use, for example, the rangeproofs described in [Max15] for Pedersen commitments, which satisfy these criteria.

### 2.1.2 Sinking Signatures

This brings us to the first new primitive needed for Mimblewimble.

**Definition 4.** *We define a* sinking signature *as the following collection of algorithms:*

- $\text{Setup}(1^\lambda)$ *outputs a keypair* $(\text{sk}, \text{pk})$*;*
- $\text{Sign}(\text{sk}, h)$ *takes a secret key* sk *and nonnegative integer "height" $h$ which is polynomial in $\lambda$, and outputs a signature $s$.*
- $\text{Verify}(\text{pk}, h, s)$ *takes a public key* pk *and signature $s$ and outputs from* $\{\text{true}, \text{false}\}$.

*satisfying the following security and correctness properties:*

- *Correctness. For all polynomial $h$,* $(\text{sk}, \text{pk}) \leftarrow \text{Setup}(1^\lambda)$*,* $s \leftarrow \text{Sign}(\text{sk}, h)$*, we have that* $\text{Verify}(\text{pk}, h, s)$ *accepts.*
- *Security. Let* $(\cdot, \text{pk}) \leftarrow \text{Setup}(1^\lambda)$*. Then given* pk *and an oracle $H$ which given $h$ returns* $\text{Sign}(\text{sk}, h)$*, no PPT algorithm $\mathscr{A}$ can produce a pair $(s, h')$ with $h' > h$ for all $h$ given to the oracle, and* $\text{Verify}(\text{pk}, h', s)$ *accepts. (Except with negligible probability.)*

The name "sinking signature" is motivated by the fact that given a signature $s$ on height $h$, it may be possible for a forger to create a signature $s'$ on height $h'$ with the same public key and $h' \leq h$, thus "decreasing the height" of the signature. We will use this feature later to aid scalability for a Mimblewimble chain.

**Definition 5.** *We say a sinking signature is* aggregatable *or* summable *if given a a linear combination* pk *of* $\text{pk}_i$ *values computed from* $\text{Setup}(1^\lambda)$*, the same linear combination of* $s_i \leftarrow \text{Sign}(\text{sk}_i, h)$ *(for fixed $h$) it is possible to compute a signature $s$ such that* $\text{Verify}(\text{pk}, h, s)$ *accepts.*

*For a summable sinking signature, we generalize the above security game to allow the adversary $\mathscr{A}$ to play polynomially many times in parallel and win with an arbitrary linear combination of its received public keys. (It must also provide the coefficients of the linear combination.)*

**Definition 6.** *We propose the following summable sinking signature (Poelstra, Kulkarni). Let $\mathscr{H}_2$ be a random oracle hash with values in $\mathscr{G}_2$.*

- $\text{Setup}(1^\lambda)$ *chooses a uniformly random* $\text{sk} \leftarrow \mathbb{Z}_r$ *and sets* $\text{pk} = \text{sk} \cdot G$*.*
- $\text{Sign}(\text{sk}, x)$ *first computes the sequence $\{x_0, \ldots, x_n\}$ where $x_0 = x$ and $x_{i+1}$ is obtained by subtracting from $x_i$ the largest power of 2 that divides it (*i.e.*, by clearing its least significant 1 bit). We observe that $x_n = 0$ and that $n$ is one plus the number of one bits in $x$ and is therefore $O(\log_2 x)$.*

  *Finally, it outputs* $s = \{\text{sk} \cdot \mathscr{H}_2(x_i\}_{i=0}^n$*.*
- $\text{Verify}(\text{pk}, x, \{s_i\})$ *computes $S$ as the sum of all $s_i$, computes $H = \sum_{i=0}^n \mathscr{H}_2(x_i)$, and checks that $e(G, S) = e(\text{pk}, H)$.*

Observe that the verification step uses only the sum $S$ of the elements of the signature. However, the extra data will be useful later, so we state it here so we can prove the scheme is secure with it.

Correctness and summability of the scheme are immediate.

**Theorem 1.** *This is a secure summable sinking signature, in the following sense: if an adversary $\mathscr{A}$ exists which wins the game described in Definition 5, a simulator $\mathscr{B}$ exists which can solve the computational co-Diffie-Hellman (co-CDH) problem for $(\mathscr{G}_2, \mathscr{G}_1)$, given oracle access to $\mathscr{A}$.*

*Proof.* $\mathscr{B}$ answers random oracle queries to $H$ and works in the following way. (Recall that the game in Definition 5 is the game from Definition 4 played in parallel.)

We suppose without loss that before making signature queries on any height $x$, the adversary first all random oracle queries needed to verify such a signature; similarly before producing a forgery on height $x^*$ it makes the required queries. We then suppose that in total it requests at most $q_p$ public keys and makes at most $q_h$ random oracle queries.

1. $\mathscr{B}$ receives a co-CDH challenge $(G, P, Q)$ from its challenger, where $G, P \in \mathscr{G}_1$, $Q \in \mathscr{G}_2$, and the goal is to produce $R \in \mathscr{G}_2$ such that $\hat{e}(P, Q) = \hat{e}(G, R)$.

2. $\mathscr{B}$ responds to the $i$th public key request by generating a uniformly random keypair $(x_i, P_i)$ and replies with $P_i + P$.

3. $\mathscr{B}$ responds to the $j$th random oracle query by generating a uniformly random keypair $(y_j, H_j)$ except that $H_{j^*} = Q$ for $j^* \xleftarrow{\$} \{1, \ldots, q_h\}$, and replying with $H_i$.

4. $\mathscr{B}$ responds to the $k$th signature query on height $h^k$ and pubkey $P^k$ as follows: it computes the sequence $\{h_n^k\}$ and checks if any of the $H(h_n^k)$ values is $Q$; if so, it aborts.

   Otherwise, for each $i\{0, \ldots, n\}$ it knows a $z_i$ such that $H(h_i^k) = z_i G$ and produces $s = \{z_i P\}_i$.

5. Finally, $\mathscr{A}$ wins by producing a forgery consisting of coefficients $\{c_i\}_{i=1}^m$ with not all $c_i$ zero, a pubkey $P^* = \sum_{i=1}^m c_i(P_i + P)$, a height $h^*$ greater than any $h^*$ thus queried on, and a signature $s^* = \{S_i\}$ such that $\sum_{i=0}^{n^*} e(G, S_i) = \sum_{i=0}^{n^*} e(P^*, H(h_i^*))$.

6. If some $H(h_{i^*}^*) = Q$, then for the above sum to hold, writing $x^*$ for $P^* = x^* G$, we must have

$$\begin{aligned}
\sum_{i=0}^{n^*} S_i &= \sum_{i=0}^{n^*} y_i^* P^* && \text{for } y_i^* \text{ such that } H(h_i^*) = y_i^* G \\
&= \sum_{i=0}^{n^*} \sum_{j=1}^{m} [c_j x_j y_i^* G + c_j y_i^* P] \\
&= \sum_{i=0}^{n^*} \sum_{j=1}^{m} [c_j x_j H(h_i^*) + c_j y_i^* P] \\
&= \sum_{j=1}^{m} c_j R + \sum_{\substack{i=0 \\ i \neq i^*}}^{n^*} \sum_{j=1}^{m} [c_j x_j H(h_i^*) + c_j y_i^* P]
\end{aligned}$$

   where $R$ is the solution to $\mathscr{B}$'s CDH challenge, and every other term is computable by $\mathscr{B}$.

To complete the proof, we must show that we abort with probability bounded away from 1, and that our win condition occurs with nonneglible probability. We observe that we only abort if the attacker asks for a signature that requires $Q$ be used, and we win if $Q$ is used in the attacker's forgery.

*Both* occur if $H(h^*) = Q$, which has probability $1/q_h$, which completes the proof. □

## 2.2 Mimblewimble Primitives

**Definition 7.** *A* Mimblewimble transaction *is the following data:*

- *A list of homomorphic commitments termed the* inputs*, with attached rangeproofs. Alternately, inputs may be given as explicit amounts, in which case they are treated as homomorphic commitments to the given amount with zero blinding factor.*
- *A list of homomorphic commitments termed the* outputs*, with attached rangeproofs.*
- *A blockheight x.*
- *An* excess *commitment to zero, with a summable sinking signature on blockheight x with this as pubkey.*

*We make the further restriction on transactions that every input within a transaction be unique.*

**Definition 8.** *We define the* sum *of a transaction to be its outputs minus its inputs, plus its excess. If*

- *the sum is zero*[4]*, and*
- *the rangeproofs and sinking signature are valid*

*then we say the transaction is* valid*.*

**Definition 9.** *We define the* canonical form *of a transaction $T$ as the transaction equal to $T$ except if any input is equal to an output, both are removed.*

*Notice that the canonical form of any transaction has equal sum to the original transaction; in particular a transaction is valid if and only if its canonical form is valid.*

We observe that all valid transactions are noninflationary; the total output value must be equal to the total input value.

**Definition 10.** *Given a finite set of transactions $\{T_i\}$ with pairwise disjoint input sets, we define the* cut-through *of $\{T_i\}$ as the canonical form of the union of all $T_i$'s.*

Next, we define some terms that will allow us to treat Mimblewimble transactions as mechanisms for the transfer of value within a blockchain.

**Definition 11.** *(Ownership.) We say that a party $S$* owns *a set of transaction outputs if she knows the opening of the sum of the outputs.*

[4] By zero we mean the homomorphic commitment which commits to zero with zero blinding factor.

**Definition 12.** *(Sending n coins.) To send n coins from S to R, S produces a transaction: chooses inputs, creates uniformly random change output(s) and a uniformly random excess, whose sum is a commitment to n. S sends this to R along with the opening information of the sum.*

**Definition 13.** *(Receiving n coins.) To receive n coins, R receives a transaction $T'$ which sums to a commitment of $m \geq n$ coins, along with opening information of this sum, and completes it to a valid transaction $T$ by the following process:*

1. *R first produces uniformly random outputs whose total committed value is n, and adds these to the transaction.*
2. *If $m = n$, R negates the sum of this transaction and adds it to the excess of the transaction, so that the total sum is now zero. (It also computes a summable sinking signature for the amount added, and adds this to the original signature, so that the final excess has a valid signature on it.)*

   *Otherwise the transaction now has sum committing to $m - n$, so R adds a uniformly random value to excess, updates the signature, and gives the opening information of the new sum to another recipient who can take the remaining value.*

When we define a blockchain and require transaction inputs to be outputs of earlier transactions, we will show that after this process, *R* is the only party who owns any of the outputs that he added.

# 3 The Mimblewimble Payment System

## 3.1 Fundamental Theorem

**Theorem 2.** *(Fundamental Theorem of Mimblewimble) Suppose we have a binding, hiding homomorphic commitment scheme. Then no algorithm $\mathscr{A}$ can win at the following game against a challenger $\mathscr{C}$ except with negligible probability:*

1. *(Setup.) $\mathscr{C}$ computes a finite list L of uniformly random homomorphic commitments. $\mathscr{C}$ sends L to $\mathscr{A}$.*
2. *(Challenge.) At most polynomially many times, $\mathscr{A}$ selects some (integer) linear combination $T_i$ of L and requests the opening of this combination from $\mathscr{C}$. $\mathscr{C}$ obliges.*
3. *(Forgery.) $\mathscr{A}$ then chooses a new linear combination $T$ which is not a linear combination of $\{T_i\}$ and reveals the opening information of $T$.*

*Proof.* Consider the lattice $\Lambda$ of formal linear combinations generated by $L$, that is, the set $\{\sum_{A \in L} b_A A : b_A \in \mathbb{Z}\}$. Consider the quotient lattice $\Lambda/\Gamma$ where $\Gamma$ is the sublattice of $\Lambda$ generated by the queries $\{T_i\}$. We may consider every element of $\Lambda/\Gamma$ to be a homomorphic commitment by using some canonical representative. In particular, every element of $\Lambda/\Gamma$ is a homomorphic commitment which satisfies the same hiding/blinding properties that the original scheme did.

Then the projection of every $\{T_i\}$ into $\Lambda/\Gamma$ is zero, and the projection of $T$ is nonzero. Therefore $\mathscr{A}$ has learned no information about the projection of $T$ from its queries; however, if $\mathscr{A}$ knows the opening of $T$ then it also knows the opening of the projection of $T$, contradicting the hiding property of the homomorphic commitment scheme. □

## 3.2 The Blockchain

Mimblewimble consists of a *blockchain*, which is a weighted-vertex directed rooted tree of *blocks* (defined below) for which the highest-weighted path is termed the *consensus history*.

**Definition 14.** *We define a* Mimblewimble block *as the following data:*

- *A canonical transaction, whose inputs are of one of the two forms:*
  - *A reference to an output of the transaction in an earlier block; or*
  - *An explicit (*i.e. *with zero blinding factor) input restricted by rules above those given in this paper*[5].
- *A* block header*, which has a binding commitment to earlier blocks in the chain, termed* back-links*; a commitment to the current UTXO set after this block's transaction has taken effect; and the transaction included in this block.*

*If the block's transaction is valid, we say the block is* valid*.*

In Section 3.3, we will describe how blocks are weighted, and which previous blocks specifically should be linked to. For now we may use Definition 14 as our definition of validity, though note that there will be additional requirements on the commitments.

**Definition 15.** *We define the* consensus chain state *of a Mimblewimble blockchain as the cut-through of all transactions on the consensus history.*

*If the consensus chain state is valid, we call the blockchain* valid*.*

Note that for a blockchain to be valid, it is not required that all blocks be valid, only that the entire chain sum to a valid transaction.

Next, we prove that Mimblewimble is a sound payment system, in the sense of the following two theorems.

**Theorem 3.** *(No inflation.) The total value committed by the outputs of the consensus chainstate of a valid blockchain is equal to the value of the explicit inputs in each block.*

*Proof.* By hypothesis the consensus chain state, which is the canonical form of the cut-through of all transactions, is valid. Since every non-explicit input of every block is the output of a previous transaction, it does not appear in this canonical form. Therefore the only inputs of the chain state are the explicit ones. □

[5] A typical rule would be that each block can have only a single *coinbase input* of fixed value.

**Lemma 1.** *(Unique ownership.) Suppose that all outputs of a transaction were created by receiving coins as in Definition 13 or sending as in Definition 12, so that all blinding factors are kept secret. Then for every subset of outputs in which not all have not been sent, the only owner of that subset is the person who created all its outputs. (In particular, if the subset contains outputs created by different parties, then that subset has no owner.)*

*Proof.* Let $O$ be an output in the subset which has not been sent. Then the only combination of outputs containing $O$ whose commitment included $O$ that may have been revealed also included some uniformly random excess $E$ which was chosen when $O$ was created.

Further no other sum containing $E$ was ever revealed, so that any combination including $O$ but not $E$ is *not* a linear combination of combinations whose opening information has been revealed. The subset in question does not contain $E$, since $E$ is an excess not an output. Therefore by Theorem 2 nobody knows the opening information of the subset except the person who created $O$. □

**Theorem 4.** *(No theft.) Consider a valid blockchain. An output $x$ created as in Definition 13 cannot be removed from the consensus chain state without replacing the block in which it appeared,* i.e.*, forming a higher-weighted valid blockchain not containing this block, except by parties who (collectively) own a set $U$ of outputs containing $x$.*

*Proof.* Suppose otherwise; then there exists a higher-weighted chain containing the block $B$ in which $x$ appeared, but for which the consensus chain state does not contain $x$.

Consider the transaction $T$ which is the canonical form of the cut-through of the first block after $B$ to the tip of the chain. (Note that $T$ may not be valid; we know only that the full chain states are valid.)

Then the outputs of $T$ are a subset of the outputs of the new chain state; in particular they contain rangeproofs which are proofs of knowledge of the openings of the outputs. Similarly, the excess value is also known. We conclude that the parties who created these blocks (and therefore $T$) know the openings of all outputs and sum of the excess value, and therefore own the set of all inputs of $T$.

However, the inputs of $T$ form a set of outputs containing $x$, completing the proof. □

## 3.3 Consensus

Mimblewimble uses a hashcash[Bac02] style blockchain in which every block in a blockchain is labelled by a weight called its *difficulty*. A blockchain is valid if every block of difficulty $D$ has a header which hashes into a range of size $1/D$ of the total space of hashes. We define a directed edge from block $A$ to $B$ iff $A$ commits to $B$ in its header, and require each block commit to its unique *parent*.

We can then refine our definition in Section 3.2 of *consensus history* as the highest-weighted path terminating at the root. This is the same as Bitcoin's design.

### 3.3.1 Block Headers

However, in we also define a *second* graph structure on blocks as vertices, called the *compact blockchain*. We define the compact blockchain iteratively as follows.

1. The genesis block is in the compact blockchain.

2. The first block after the genesis is added to the compact blockchain, and is assigned *effective difficulty* equal to its difficulty.

3. Each new block is added to the tip of the compact blockchain, and may cause blocks to be removed from the compact chain as follows:

   (a) First, its effective difficulty is calculated as follows. Consider the "maximum possible difficulty" $M$ of the block, which is the size of the hash space divided by the hash of the block. (This may be larger than the actual difficulty.)

   The effective difficulty is determined by starting with the block's difficulty, then adding the effective difficulties of as many consecutive blocks as possible, starting from the tip, so that the total is less than or equal to $M$.

   (b) All blocks whose effective difficulty was used in the above calculation, except the new block itself, are dropped from the compact chain.

The compact blockchain is encoded in the real blockchain by having every block commit to a merkle sum tree (with effective difficulty the quantity being summed) of all blocks in the current compact chain.

We next prove several theorems to give an intuition of the properties of the compact blockchain.

**Lemma 2.** *The expected work required to produce a block with effective difficulty D is equivalent to computing D hashes; similarly to produce several blocks with total effective difficulty $D'$ one must do expected work of computing $D'$ hashes.*

*Proof.* This is immediate in the random oracle model. □

**Theorem 5.** *The expected amount of work to replace a block B in the compact chain (*i.e. *produce a blockchain of greater or equal total difficulty whose compact chain does not contain B) is greater than or equal to the work needed to replace B, its parent in the non-compact chain, its parent, and so on, up to but not including B's parent in the compact chain.*

*Proof.* This follows immediately from Lemma 2 and the fact that the effective difficulty of every block is defined to be greater than or equal to the sum of the difficulty of the skipped blocks. □

**Corallary 1.** *The expected work required to produce a compact blockchain is at least as large as the expected work required to produce a full chain containing the same blocks.*

**Theorem 6.** *Assuming constant difficulty, given a blockchain of length N, the expected length of the compact chain will be $O(\log N)$.*

*Proof.* The compact chain has been defined such that the proof in [BCD+14, Appendix A] still holds. We summarize it here:

1. First, consider starting from the tip and scanning backward until we find a block that can skip all remaining blocks back to the genesis. By construction this block will be in the compact chain. The probability that such a block exists within the first $x$ blocks we check is

$$1 - \prod_{i=1}^{x} \frac{N-i}{N-i+1} = 1 - \frac{N-x}{N} = \frac{x}{N}$$

and the expectation of this over all $1 \leq x \leq N$ is $\frac{N+1}{2}$, *i.e.* we expect the chain length to be halved by this one skip.

2. Inductively, we can scan back from the tip until we find a block that skips back to the block in the previous step. This halves (in expectation) the remaining chain, and so on.

The number of times we repeat this process until we have no more blocks to skip is the length of the compact chain, since each step added one more block to the compact chain, and since each step halved the number of remaining blocks, we see that there are only logarithmically many steps. □

We observe that small variations in difficulty do not affect the character of this proof, and therefore the constant-difficulty case can be considered a good approximation to the real-world situation.

**Theorem 7.** *Given a blockchain $B$ (for the purpose of this theorem, we considering $B$ to be only the most-work path), there is exactly one compact chain $C \subseteq B$. Further, verifiers can determine that that $C$ is the compact chain given only the blocks of $C$ and the opening of each block's commitment to the previous in the compact chain. Finally, no blocks in $B \setminus C$ will appear in the compact chain of any extension of $B$ (i.e. once a block is dropped it may be forgotten forever).*

*Proof.* By construction, a compact chain $C$ exists which is a subset of $B$. Suppose that some other compact chain $C' \neq C$ of $B$ also exists. Let $C_{\leftarrow}$ be the longest path from the tip contained in both $C'$ and $C$ (since the tip itself is in both $C'$ and $C$ by construction, this is nonempty), and let $\beta$ be the deepest block of $C_{\leftarrow}$.

Now, the block preceding $\beta$ must differ in $C$ and $C$; however, this is impossible since $\beta$ commits to an ordered Merkle sum tree of previous blocks in the compact chain, the "preceding block" must be the first one that $\beta$'s hash is not small enough to skip, and this is uniquely specified by the shape of the Merkle sum tree. We conclude that $C' = C$.

Next, we argue that when $B$ is extended, no blocks of $B \setminus C$ are added to the compact chain. But this is immediate, since by construction only new blocks are ever added to a compact chain. □

### 3.3.2 Proven and Expected Work

However, while the expected work can be computed to be the same, the the compact chain **does not**, in general, prove as much work as the full chain. Here by "proving an amount of work" we mean that a prover who does less than this amount of work has negligible chance of producing the chain. For example, consider a blockchain of total difficulty $D$ across $n$ blocks, whose compact chain has $\log n$ blocks.

Suppose an attacker attempts to produce this chain in $\varepsilon D$ work, where $0 < \varepsilon < 1$. Then this requires, on average, that each individual block be produced in $\varepsilon$ the expected time. Each block's

production time is an independent variable, so the Chernoff bound lets us approximate this more precisely: if $\varepsilon < 1$ then the probability decays exponentially with the number of blocks in the chain.

For the full chain this means probability $O(\exp[(1-\varepsilon)n])$ (exponential in $n$); for the compact chain $O(\exp[(1-\varepsilon)\log n])$ or $O(n^{1-\varepsilon})$ (sublinear in $n$). In fact, the extreme case is even worse: a compact chain may consist of only a single block which has difficulty $D$, in which case the probability is simply $O(\exp(1-\varepsilon))$. This means an attacker willing to expend some fixed percentage of the total chain work has the same probability of successfully rewriting the chain regardless of the length of the chain.

To be conservative, we conclude that **compact chains, as described in this paper, prove no work.**[6].

So what good are they? In Mimblewimble, we expect verifiers of a chain to demand all blocks from the most recent two months, say, and a compact chain from there to the start (in Section 3.3.3 we will see how full verification can proceed using only a compact chain). The resulting composite will:

- be forgeable with expected work equal to the entire chain's work; but
- only *prove* the most recent two months of work

Unlike Bitcoin, where the expected work to forge a blockchain is the same (asymptotically) to its proven work, in Mimblewimble these quantities are different. The expected work affects incentives: rational actors will choose to extend the most-work chain rather than attempting forgeries, just as in Bitcoin; on the other hand, the proven work affects verifiers' certainty about the state of the world: they know at least two months worth of work has been done to produce the chain they see, that it was not an accident, and that if it is a forgery it was a very expensive one that cannot be reliably repeated.

### 3.3.3 Sinking Signatures and Compact Chains

In this section, we describe how sinking signatures interact with compact chains, and in particular we find that it is possible to do a full Mimblewimble verification with only a compact chain. We introduce a notion of *compact validity* of blocks which which is weaker than validity as described in in Definition 14. Nonetheless, we preserve the trust model of Section 1.2.

To do this, we modify the Merkle sum tree of previous blocks and their effective difficulty to sum not only difficulty, but (a) the excess of the blocks' transactions (Definition 7), (b) the sinking signatures on this excess. The excess values are summed in the obvious way, added as points, but the sinking signatures are more complicated.

Rather than directly adding the signatures, we combine *sets* of signatures from each child at every node of the Merkle tree, in the following fashion: for any blockheight $x$, let $\{x_n\}$ denote the decreasing sequence defined in Definition 6. Then the signature set on a node is the union of the signature set of its children, except that whenever two heights $x < y$ appear in the set such that

[6]This says nothing about the compact SPV proofs described in, e.g. [KLS16], which put lower bounds on the length of compact proofs in order to upper-bound the probability a less-than-expected-work attacker can succeed. We cannot take this approach because we are using these proofs in consensus code and therefore need Theorem 7.

$\{x_n\} \subseteq \{y_n\}$, then both signatures are dropped and replaced by the signature on $\{x_n\}$ obtained by adding the corresponding components of the original signature.

Therefore for a block at height $h$, the root of the Merkle sum tree committing to its ancestor blocks will have $O(\log h)$ sinking signatures, one for every power of 2 between 0 and $h$.

With this structure in place, we are able to define validity of a block.

**Definition 16.** *A block is* compact valid *if*

- *It is valid in the sense of Definition 14.*
- *Its Merkle sum tree commits to the deepest block allowable under the rules of Section 3.3.1.*
- *This commitment is valid, in the sense that all the summing rules are obeyed.*
- *The above commitment will be a Merkle proof consisting of a path from the commitment to the Merkle root of the form $\{c_i, c_i'\}$ where $c_0$ is a direct commitment to the block; for all $i$ $c_i'$ is the sibling in the Merkle tree of $c_i$; and $c_{i+1}$ is a commitment to $\{c_i, c_i'\}$.*

  *We require the first $c_i'$ that is a* right sibling *in the tree to have a valid set of sinking signatures on it. (If there is no such $c_i'$ then this block is skipping back to the genesis, so we instead require that all the signatures at the root of the tree are correct.)*

**Definition 17.** *A block is* fully valid *if it is compact valid and*

- *It commits to its immediate predecessor in the full chain (and the excess/signature committed to in the tree match the previous block).*
- *It has a valid commitment to the current UTXO set at the time of its creation.*

(The latter condition allows the UTXO set to be verified using only the compact chain; it also prevents consensus attacks whereby users are given the same chain but differing UTXO sets that it commits to. The former condition ensures that compact blocks' contents don't differ from full blocks' contents, as long as full verifiers are watching.) (And if nobody is watching, it's a moot point anyway.)

The conditions of Definition 16 are very technical. We can summarize them simply as follows: *whenever a block at height h skips back to $h'$, we sink the signatures of every intervening block to the lowest height greater than $h'$ possible, then aggregate all the signatures that wound up on the same height*. This mess of Merkle sum trees is only to show how this condition can be checked by a verifier given only polylogarithmically much data.

We argue that this design preserves the trust model. In particular:

**Theorem 8.** *No transaction can be removed without (doing as much work as) rewriting the block it appeared in.*

*Proof.* Every transaction has a sinking signature on the height $h$ of the block it appears in. When a block is removed from the compact chain, the above construction may cause this signature to only be verified after being sunk to some lower height $h'$.

However, since $h'$ is always guaranteed to lay between the same two blocks in the compact chain that $h$ does, this signature cannot be removed except be rewriting the more recent of these two blocks. However, by construction, this has expected work greater than or equal to rewriting the block that the transaction originally appeared in. □

**Theorem 9.** *The commitments required by Definition 16 take $O(\log^3)$ space in the height of the full chain.*

*Proof.* The commitment contains logarithmically many nodes from the Merkle tree, each of which contain logarithmically many sinking signatures, each of them are logarithmic in size. □

# 4 Extensions and Future Research

**Multisignature Outputs.** We observe that CT rangeproofs can be produced interactively in the same ways that Schnorr signatures can to produce multisignature outputs. Similarly the sinking signatures can be trivially produced in a multiparty way. So support for multiparty signatures, while not addressed in this article, is simply a matter of wallet support and requires no further changes to the system.

**Payment channels.** Bitcoin's script system supports off-chain *payment channels*, which are used by the Lightning network[PD16] to support unlimited transaction capacity in constant blockchain space. A scaling-focused blockchain design such as Mimblewimble ought to support such a scheme.

It is an open problem to produce payment channels that are as ergonomic and flexible as those in Bitcoin, but to show that this ought to be possible, here is an example of a primitive Mimblewimble payment channel. Suppose that a party $A$ wants to send a series of payments to party $B$ of maximum value $V$.

1. First $A$ creates a spend of $V$ coins to a multisignature output requiring both $A$'s and $B$'s signature to sign. $A$ "timelocks" this by taking the highest 0 bit $i$ in the current blockheight $h$, then asking $B$ to sign a transaction with height $h+2^i$ returning the coins to $A$. Only after receiving this signature, $A$ publishes the spend.

   The signing signature construction ensures that such a refund transaction cannot be spent in any block between $h$ and $h+2^i$. On the other hand, this means that if $A$ wants a locktime of $2^i$ blocks he must wait for a blockheight that is a multiple of $2^{i+1}$ to create it.

2. Now that all $V$ coins are in a multisignature output requiring both $A$'s and $B$'s signatures to spend (with the coins going back to $A$ after $2^i$ blocks in case of stall), $A$ can send an amount $v$ to $B$ by signing a transaction from this output sending $v$ to $B$ and $(V-n)$ to $A$.

3. To increase the value $v$, $A$ simply signs a new transaction with the new value $v$. $B$ will not sign and publish until the channel is near-closing, at which point $B$ can publish one transaction taking the whole value $v$, even if it was actually produced over the course of many interactions.

**Sidechain support.** If Mimblewimble blocks commit to a structure containing all peg-in transactions and all peg-out transactions (in the same way they commit to the UTXO set, except these structures will never shrink), then it is possible for Mimblewimble to be implemented as a pegged sidechain. This would provide a tremendous scaling benefit to its parent chain, since most blockchain transactions are of the simple transfer-of-value sort that Mimblewimble supports, and also reduce the risk to users of Mimblewimble from quantum computers, since it is easy to move coins off of a sidechain.

On a technical level, both peg-ins and peg-outs may look like transaction excess values which, instead of signing blockheights, sign an output on the destination chain. Then verifiers add this excess plus as $\pm vH$ to the total utxoset value (where $v$ is the value of the output).

Working out the details of this, and arguing security, is left as future research, but it appears on a high-level that there are no open problems.

**Quantum resistance.** Since Mimblewimble depends on the discrete logarithm problem for security against theft and inflation, it is highly susceptible to attacks by quantum computers. Finding quantum-secure analogues to the primitives used in this paper would allow a quantum-secure Mimblewimble with the same asymptotic scaling properties of the original. Some research in this direction includes:

- Pedersen commitments can be replaced by a quantum analogue such as [CDG+15]. Note that these commitments are not homomorphic in the strong sense of allowing arbitrarily many commitments to be added, since after a fixed noise threshold the commitments will no longer open — however, this is not a problem since Mimblewimble only requires that (unmerged) transactions sum to (a non-hiding) commitment to zero.
- Sinking signatures can likely be replaced by a LWE-based candidate, or a variation of the NIZK proof-of-openings given in the above paper. The only interesting property required of these is that signatures on same value can be added to form multisignatures, which should be algebraically easy to obtain.
- The author is unaware of any quantum-secure rangeproofs.
- The blockchain commitments, being based on hashes, are already quantum secure, and the compact chain arguments go through unchanged.

# 5 Acknowledgements

The author would like to thank

- Avi Kulkarni for developing the sinking signature construction described in the paper (and breaking the author's original construction).
- Gregory Sanders for asking probing questions about Mimblewimble's guarantees until a clear picture of its consensus structure appeared.

# References


[Bac02] A. Back, *Hashcash — a denial of service counter-measure*, 2002, http://hashcash.org/papers/hashcash.pdf.

[BCD+14] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, *Enabling blockchain innovations with pegged sidechains*, 2014, https://www.blockstream.com/sidechains.pdf.

[CDG+15] D. Cabarcas, D. Demirel, F. Göpfert, J. Lancrenon, and T. Wunderer, *An unconditionally hiding and long-term binding post-quantum commitment scheme*, Cryptology ePrint Archive, Report 2015/628, 2015, http://eprint.iacr.org/2015/628.

[Jed16] T.E. Jedusor, *Mimblewimble*, 2016, Defunct hidden service, http://5pdcbgndmprm4wud.onion/mimblewimble.txt. Reddit discussion at https://www.reddit.com/r/Bitcoin/comments/4vub3y/mimblewimble_noninteractive_coinjoin_and_better/.

[KLS16] A. Kiayias, N. Lamprou, and A.-P. Stouka, *Proofs of proofs of work with sublinear complexity*, Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers (Berlin, Heidelberg) (J. Clark, S. Meiklejohn, Y.A.P. Ryan, D. Wallach, M. Brenner, and K. Rohloff, eds.), Springer Berlin Heidelberg, 2016, pp. 61–78.

[Max13a] G. Maxwell, *CoinJoin: Bitcoin privacy for the real world*, 2013, BitcoinTalk post, https://bitcointalk.org/index.php?topic=279249.0.

[Max13b] ———, *Transaction cut-through*, 2013, BitcoinTalk post, https://bitcointalk.org/index.php?topic=281848.0.

[Max15] ———, *Confidential transactions*, 2015, Plain text, https://people.xiph.org/~greg/confidential_values.txt.

[Max16] ———, *The first successful zero-knowledge contingent payment*, 2016, Blog post, https://bitcoincore.org/en/2016/02/26/zero-knowledge-contingent-payments-announcement/.

[Mou13] Y. M. Mouton, *Increasing anonymity in Bitcoin ... (possible alternative to Zerocoin?)*, 2013, BitcoinTalk post, https://bitcointalk.org/index.php?topic=290971.

[Nak09] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009, https://www.bitcoin.org/bitcoin.pdf.

[Nol13] T. Nolan, *Re: Alt chains and atomic transfers*, https://bitcointalk.org/index.php?topic=193281.msg2224949#msg2224949, 2013.

[PD16] J. Poon and T. Dryja, *The bitcoin lightning network*, 2016, `https://lightning.network/lightning-network-paper.pdf`.

[Ped01] T. Pedersen, *Non-interactive and information-theoretic secure verifiable secret sharing*, Lecture Notes in Computer Science **576** (2001), 129–140.