Mimblewimble

Andrew Poelstra

2016-08-11

Abstract

This is badly incomplete and some proofs are wrong. If you have encountered this paper floating around the Internet, please disregard it. Sorry to be a tease. Full version forthcoming.

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 TODO CITE, 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 CITE CITE. The purpose of this paper is to make precise the original idea, and add further scaling improvements developed by the author.

1 Preliminaries

1.1 Cryptographic Primitives

Groups. Throughout, $\mathscr{G}_1$, $\mathscr{G}_2$ will denote elliptic curve groups adorned with an efficiently computable bilinear pairing $e : \mathscr{G}_1 \times \mathscr{G}_2 \to \mathscr{G}_T$, 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. and be generated by fixed elements G_1 and G_2, r whose discrete logarithms relative to each other are unknown (*i.e.* they are nothing-up-my-sleeve (NUMS) points. We will make computational hardness assumptions about these groups as needed. We write $q = |G|$, $\mathscr{Z}_q$ for the integers modulo q, and write $+$ for the group operation in all groups.

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

1.1.1 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* $\text{Open}(v',r',\text{Commit}(v,r))$ *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* $\text{Commit}(v,r)$ *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:* $\text{Commit} : \mathbb{Z}_q^2 \to \mathscr{G}$ *maps* (v,r) *to* $vH+rG$*, and* $\text{Open} : \mathbb{Z}_q^2 \times \mathscr{G} \to \{\text{true},\text{false}\}$ *checks that* (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. TODO cite pedersen*

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) (TODO cite) 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.

TODO cite greg's rangeproofs for pedersen commitments

1.1.2 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* (sk,pk)*;*

- $\text{Sign}(\text{sk},h)$ *takes a secret key* sk *and nonnegative integer "height" h which is polynomial in* λ*, 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 (p,h), it may be possible for a forger to create a signature (p',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* 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.)*

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

- $\text{Setup}(1^\lambda)$ chooses a uniformly random $\text{sk} \leftarrow \mathbb{Z}_n$ 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 $O(\log_2 x)$.

  Next, it sets $s = \{\text{sk} \cdot H(x_i\}_{i=0}^n$.
- $\text{Verify}(\text{pk}, x, p)$ computes P as the sum of all elements of p, $H = \sum_{i=0}^n H(x_i)$, and checks that $e(G,P) = e(\text{pk}, H)$.

Observe that the verification step uses only the sum P of the elements of the proof. 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 Diffie-Hellman (CDH) problem in* $\mathscr{G}$*, 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 CDH challenge (P,Q) from its challenger.
2. $\mathscr{A}$ responds to the ith public key request by generating a uniformly random keypair (x_i, P_i) and replies with $P_i + P$.

3. $\mathscr{A}$ responds to the jth 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. $\mathscr{A}$ responds to the kth 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, $\mathscr{B}$ 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 $\mathscr{B}$.

TODO show we don't abort (except with probability bounded away from 1), this win condition occurs with nonneglible probability, also let the challenger add its own pubkeys (as long as it declares them) (in MW it hasta declare them because any "excess" it adds is the sum of a subset of outputs) □

1.2 Mimblewimble Primitives

Definition 6. *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 7. *We define the* sum *of a transaction to be its outputs minus its inputs, plus its excess. If the sum is zero*[1]*, we say the transaction is* valid*.*

Definition 8. *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 9. *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 10. *(Ownership.) We say that a party S* owns *a set of transaction outputs if she knows the opening of the sum of the outputs.*

Definition 11. *(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 12. *(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.*

   Otherwise the transaction now has sum committing to $m - n$, so R adds a uniformly random value to excess, then 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.

2 The Mimblewimble Payment System

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

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. TODO I think this proof is totally wrong, can probabily use it to prove things that are vulnerable to Wagner's algo (tho this is not) ... need to somehow define "linear combination" in a way that excludes combinations that are not computationally accessible; similar for "lattice" and "projection". Or take a different strategy entirely.

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

Then the projection of every $\{T_i\}$ into Λ/Γ 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. □

2.1 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*. [TODO cite]

Definition 13. *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*[2].
- *A* block header*, which has a binding commitment to earlier blocks in the chain, termed* backlinks*; the transaction included in this block; the cut-through of this transaction and every previous block's transactions.*

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

In Section 2.2, we will describe how blocks are weighted, and which previous blocks specifically should be linked to. For now we will

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

Definition 14. *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. □

Lemma 1. *(Unique ownership.) Suppose that all outputs of a transaction were created by receiving coins as in Definition 12, and that all blinding factors are kept secret with the specific exception of sending coins as in Definition 11. 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 12 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. □

2.2 Consensus

Mimblewimble uses a hashcash [TODO cite] 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 2.1 of *consensus history* as the highest-weighted path terminating at the root. This is the same as Bitcoin's design.

2.2.1 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 block added to the tip of the compact blockchain 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.

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 CITE, 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. *for a given blockchain, there is exactly one compact chain, and it can be verified with log-much work. further, any block which appears in the full chain but not the compact chain, will not appear in the compact chain of any extension of the full chain (i.e. once a block is dropped it can be forgotten).*

Proof. TODO TODO □

2.2.2 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. 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 εD work, where $0 < \varepsilon < 1$. Then this requires, on average, that each individual block be produced in ε 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 $O(\exp((1-\varepsilon)n))$; for the compact chain $O(\exp((1-\varepsilon)\log n))$ or $O(n^{1-\varepsilon})$. 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**[3].

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 2.2.3 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.

TODO this is a pretty serious deviation from the bitcoin trust model, needs to be summarized in a new section 1.1 "trust model"

2.2.3 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 $\log^2$ of the total historic block data.

3 Conclusions

3.1 Scaling Properties

[3]This says nothing about the compact SPV proofs described in, e.g. CITE PROOFS OF POW, 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.