How to Mimblewimble

Jakob Abfalter

September 3, 2019

1 Introduction

This paper explains how Mimblewimble transactions look like, how they are generated and how the ledger is constructed. It is a summary of parts of an article by Fuchsbauer et al. [1].

2 Transactions

2.1 Transaction structure

- A coin in Mimblewimble is of the form $C = vH + rG, \pi$ so C is a (pedersen) commitment to the value v, with randomness r and generators H and G. π is a range proof.
- In contrast to Bitcoin there is no notion of addresses or scripts. Ownership of a coin is equivalent to the knowledge of its opening. The randomness r of the commitment now acts as the secret key for the coin.
- A transaction consists of $C = (C_1, ..., C_n)$ input coins and $C' = (C'_1, ..., C'_n)$ output coins.

A transaction is valid iff $\sum v'_i - \sum v_i = 0$, so we get the equation

$$\sum C' - \sum C = \sum v'_i * H + r'_i * G - \sum v_i * H + r_i * G$$

If we assume the transaction is valid we are left with the excess

$$E := (\sum r'_i - \sum r_i) * G$$

Knowledge of the opening of all coins and validity of the transaction implies knowledge of E.
However directly revealing the opening of E would leak too much information. (An adversary knowing the openings for input coins and all but one output coin, could calculate that opening knowing the Excess)
Therefore we prove the knowledge of E by providing a signature which validates

for E as a public key.
Coinbase transactions additionally have a supply added to the excess

$$E := (\sum r'_i - \sum r_i) * G - s * H$$

Finally a Mimblewimble transactions consists of:

$$tx = (s, C, C', K) \quad with \quad K = (\pi, E, \sigma)$$

where s is the supply, C are input coins, C' output coins, K the so called Kernel consisting of π which are range proofs for the output coins, E which is a list of excess values and σ which are signatures.

2.2 Transaction merging

An important property of the Mimblewimble protocol is that two transactions can easily be merged into one. Assume we have two transactions:

$$tx_0 = (s_0, C_0, C'_0, (\pi_0, E_0, \sigma_0))$$

$$tx_1 = (s_1, C_1, C'_1, (\pi_1, E_1, \sigma_1))$$

Then the merged transaction simply is

$$tx_m = (s_0 + s_1, C_0 \mid\mid C_1, C'_0 \mid\mid C'_1, (\pi_0 \mid\mid \pi_1, (E_0, E_1), (\sigma_0, \sigma_1)))$$

Note that if the signature scheme supports merging of signatures such as the BLS signature scheme, the two signature can be replaced by one reducing the size of the transaction.
The merged transaction is valid iff the transactions it is composed of are valid.

2.3 Cut through

Assume C appears as an ouput in tx_0 and as an input in tx_1, then we can erase C from the input and output list and the transaction will still be valid. This means everytime an output is spent it is essentially forgotten, improving privacy and yieling in space savings.

2.4 Ledger

The Mimblewimble ledger itself is a transaction of the discussed form. Initially the ledger starts out empty and transactions are added and aggregated recursively. Trivially only transactions can be added which input coins are contained in the ouput coins of the ledger.

- The supply of the ledger is the sum of the supplies of all transactions added to the ledger so far, so the total supply.
- Due to cut through the input coin list of the ledger is always empty and the output list is what is called UTXO set in Bitcoin.

3 Transaction creation protocol

3.1 Original protocol

1. Sender selects input coins C of total value $v \geq p$ (p is the value he wants to transfer to the receiver), creates change coins C' of total value $v - p$ and then sends C, C', the range proofs for C', plus the opening $(-p, k)$ of $\sum C' - \sum C$ to the receiver.
2. The receiver creates additional output coins C'' plus range proof of total value p with keys (k_i'') and computes a signature with secret key $k + \sum k_i''$ and finializes the transaction as

$$tx = (0, C, C' \mid\mid C'', (\pi, E = \sum C' + \sum C'' - \sum C, \sigma)$$

and publishes it to the ledger.

This protocol however turned out to be vulnerable. The receiver is able to spend the change coin C' by reverting the transaction, as he can compute valid range proofs and excess signatures. This would essentially give the sender his coin back, however as the sender might not have the keys for that coin anymore, the coins could be lost.
This was solved (in Grin) by engaging in a two party protocol to compute the signature such that the sender does not have to send his share of the key to the receiver.

3.2 Salvaged protocol

The authors of the paper [1] propose a slightly modified version of the protocl not vulnerable to this problem.

1. The sender constructs a full-fledged transaction spending his input coin C, creating change coins C', as well as a special output coin $C = pH + kG$ and sends the tx and the opening (p, k) of the special coin to the receiver.(Note that, unlike in the previous case, k is now independent from the keys of the coins C and C'.)
2. The receiver now creates a second transaction tx' spending the special coin, creating its own output coins C'' and aggregates tx and tx'. The receiver then publishes the merged transaction to the ledger.

The only drawback of this approach is that we two transactions kernels instead of just one, making the transaction slightly bigger.

References

[1] Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: a cryptographic investigation of mimblewimble. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 657–689. Springer, 2019.