Mimblewimble and Scriptless Scripts

Andrew Poelstra

grindelwald@wpsoftware.net

January 10, 2018

What is a Blockchain?

- For our purposes, a *blockchain* is a Merkleized linked list of commitments, called *blocks*, along with rules restricting the committed data.
- (Also, critical but irrelevant magic, there is global consensus on what this list is.)
- In Bitcoin the blocks are Merkle trees of transactions, each of which may not conflict with any other across the entire chain.

What is Mimblewimble?

- Anyone can download the blockchain, validate all the committed data, and determine the current system state, the *unspent transaction output set (utxoset)*.
- Basically every cryptocurrency uses this model, up to structure and naming of the system state.
- Mimblewimble, proposed in August 2016 by Tom Elvis Jedusor, is an alternate design where transaction data eventually becomes irrelevant and can be dropped, even for new validators.

Talk Outline

1. How are Mimblewimble transactions structured to enable this redundancy?

   Hint: they are restricted to be very simple.

2. How, despite these restrictions, can we still execute trustless multiparty cryptosystems ("smart contracts")?

Confidential Transactions and Pedersen Commitments

- Given a dollar value $v \in \mathbb{Z}/q\mathbb{Z}$, choose uniformly random $r \in \mathbb{Z}/q\mathbb{Z}$ and compute

  $$C = vH + rG$$

  where H, $G \in \mathcal{G} \simeq \mathbb{Z}/q\mathbb{Z}$ are generators of a DL-hard group.

- Attach a *rangeproof* that $v << q$, *i.e.* our amounts are in the part of $\mathbb{Z}/q\mathbb{Z}$ that basically acts like $\mathbb{Z}^+$.

- Replace all the amounts in a Bitcoin transaction with Pedersen commitments; verifiers check for each transaction that

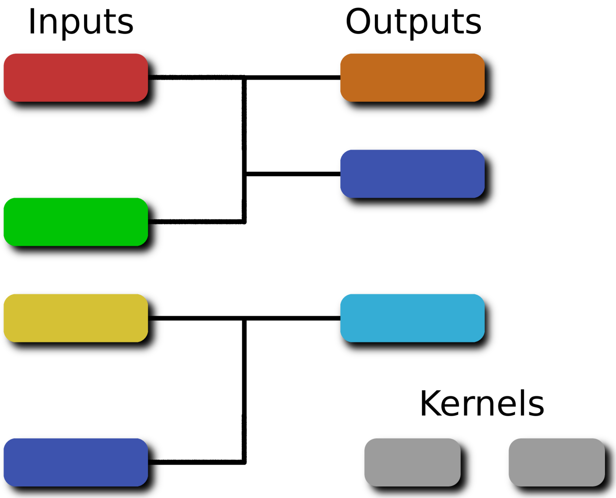
  $$\sum_{C \in \text{inputs}} C \quad = \quad \sum_{C \in \text{outputs}} C$$

Mimblewimble: Transactions

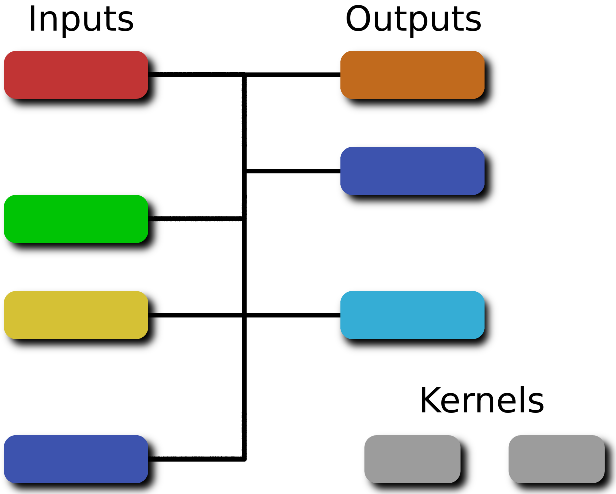
- Observe that the blinding factors r in the input commitments must sum to the blinding factors in the output commitments.
- Therefore it is impossible to construct a transaction without knowing the sum of its inputs' blinding factors, each of which should be secret.
- Mimblewimble: drop all other forms of authentication and just do this.

- This almost works, except that parties within the same transaction would learn about each others' secret blinding factors r. (Sum of one party's blinding factors equals the sum of the other's.)
- By adding an unspendable 0-valued output to each transaction, called a *kernel*, multiple parties can produce a transaction together without anyone learning each others' secret r values.
- Each participant i chooses a blinding factor ρ_i and sets the kernel commitment to $K = \sum_i \rho_i G$. They produce a multisignature with this key to authenticate the transaction and prove that K is 0-valued.

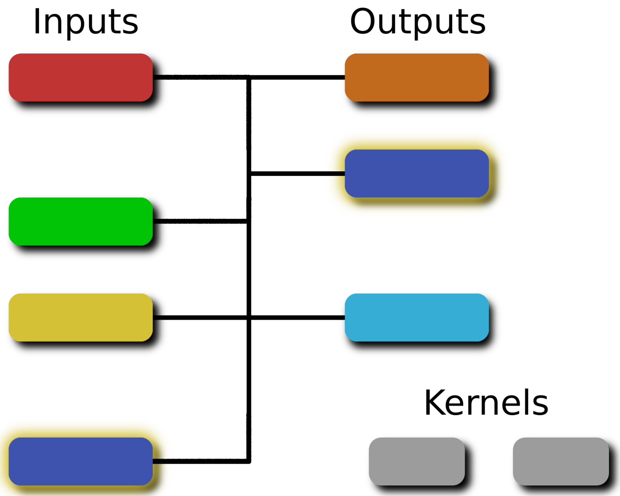
Mimblewimble in Pictures



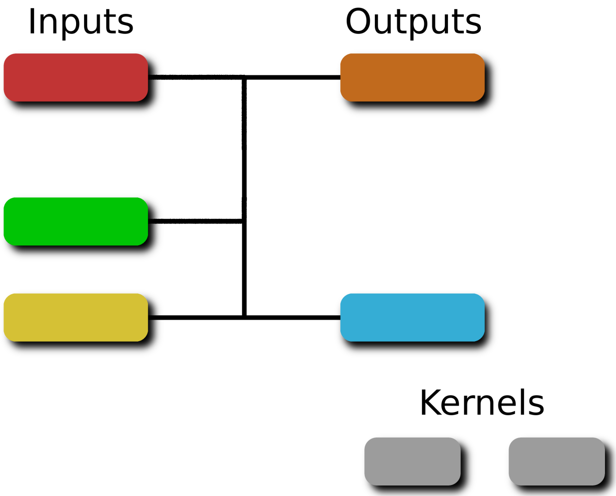
Mimblewimble in Pictures



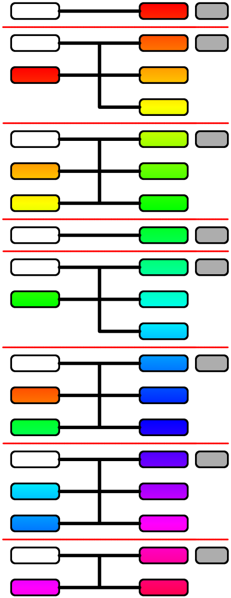
Mimblewimble in Pictures



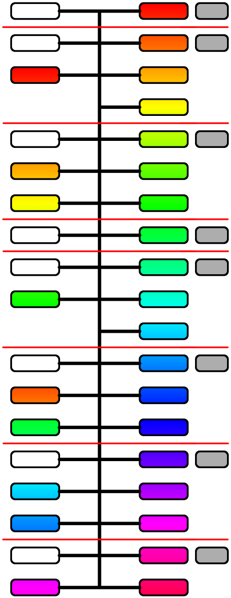
Mimblewimble in Pictures



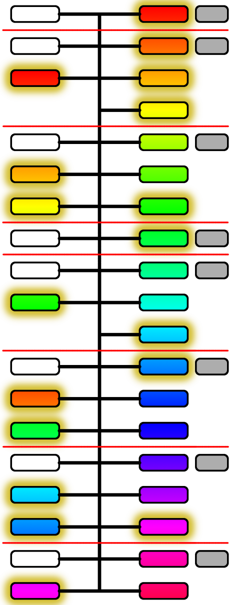
Mimblewimble in Pictures



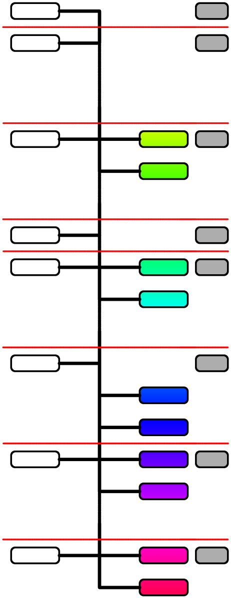
Mimblewimble in Pictures



Mimblewimble in Pictures



Mimblewimble in Pictures



Mimblewimble Scaling: Real Numbers

- In Bitcoin there are 150 million transactions with about 400 million outputs, 65 million of which are unspent.
- This takes about 180Gb of space on disk today; with CT this would increase by another 270Gb.
- MimbleWimble gives us CT and requires storing: 18Gb of transaction kernels, headers etc.; 2Gb of unspent outputs, and 45Gb of UTXO rangeproofs.

“Scriptless Scripts”

- Scriptless scripts: magicking digital signatures so that they can only be created by faithful execution of a smart contract.
- Limited in power, but not nearly as much as you might expect.
- Mimblewimble, having no permanent data except kernels and their signatures, supports only scriptless scripts, But anything that supports Schnorr signatures will support scriptless scripts.

Why use Scriptless Scripts?

- Bitcoin (and Ethereum, etc.) uses a scripting language to describe smart contracts and enforce their execution.
- These scripts must be downloaded, parsed, validated by all full nodes on the network. Can't be compressed or aggregated.
- The details of the script are visible forever, compromising privacy and fungibility.
- With scriptless scripts, the only visible things are public keys (i.e. uniformly random curvepoints) and digital signatures.

Schnorr Signatures Support Scriptless Scripts

- Basic Schnorr multisignature: signers have keypairs (x_i, P_i) with $P_i = x_i G$.
- Agree on a message, compute uniformly random $R_i = k_i G$, and exchange R_i.
- Each computes $R = \sum_i R_i$, $P = \sum_i P_i$, $e = H(P \| R \| m)$, and $s_i = k_i + e x_i$.
- Signature is (s, R) with $s = \sum_i s_i$. Validates as $sG = P + eR$.
- (Here we ignore key cancellation attacks etc. Be careful!)

Schnorr multi-Signatures are Scriptless Scripts

- Observe that this multisignature is already a scriptless script: the signing parties agree on a set $\{P_i\}$ of keys, but blockchain validators see only the sum P and don't care about the details.
- Can be generalized to m-of-n by linear secret sharing.
- In general, scriptless scripts will derive their power from these signatures being (verifiably) linear in all secret inputs.

Adaptor Signatures

- Consider the Schnorr multisignature construction, modified such that the first party generates $T_1 = t_1 G$. In place of R_1 it passes $R_1 + T_1$ to the other parties. Alongside s_1 it passes T_1. Nothing else changes

- We call the set $(T_1, T_1 + R_1, s_1)$ an *adaptor signature*.

- The final signature (s, R) isn't valid, but $(s + t_1, R)$ is.

- Before signing, the other parti(es) verify $s_1 G = R_1 + eP_1$, and therefore that knowledge of t_1 will be equivalent to knowledge of a valid signature.

Features of Adaptor Signatures

- By attaching auxiliary proofs to T_1 to ensure t_1 is some necessary data for a separate protocol, arbitrary steps of arbitrary protocols can be made equivalent to signature production.

- In a blockchain context, this means parties can be trustlessly paid for continued honest participation.

- In particular, by using the same T_1 in multiple adaptor signatures it is possible to make arbitrary sets of signatures atomic, as we will see in the next example. Extremely cheap.

- After a signature hits the chain, anyone can make up a T_1 and compute a corresponding "adaptor signature" for it, so such schemes are deniable/private.

Example: Atomic (Cross-chain) Swaps

- Suppose Alice wants to trade 10 A-coins for 5 of Bob's B-coins.
- On their respective chains, each moves the coins to outputs that can only be spent by a 2-of-2 multisignature with both Alice and Bob.
- They do sign the multisignature protocols in parallel, except that in both cases Bob gives Alice adaptor signatures using *the same* T_1.
- Bob replaces one of the signatures (s, R) with $(s + t_1, R)$ and publishes it, to take his coins. Alice sees this, learns t_1, then does the same thing on the other chain to take her coins.

Open Problems

- Quantum-resistant analogues to all this
- Scriptless scripts with more than 2 parties
- Formalizing/understanding limits of scriptless scripts

Thank You

Andrew Poelstra <grindelwald@wpsoftware.net>