Scriptless Scripts With Mimblewimble

Andrew Poelstra

Research Director, Blockstream
grincon@wpsoftware.net

January 28, 2018

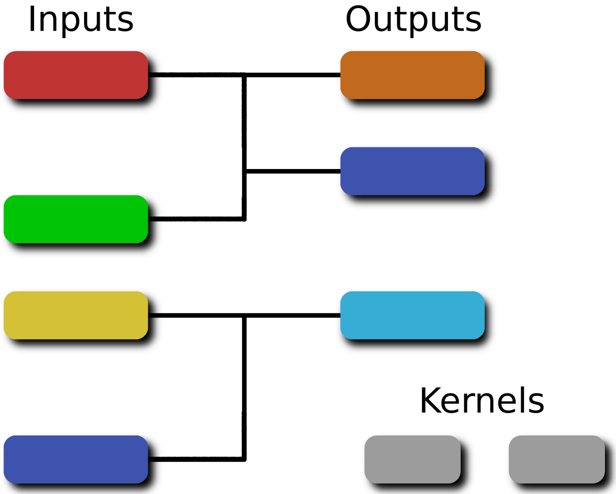
Mimblewimble and Scriptless Scripts

- Mimblewimble, proposed in 2016 by Tom Elvis Jedusor
- No scripts, only signatures.
- "What script support is possible? We would need to translate script operations into some sort of discrete logarithm information."

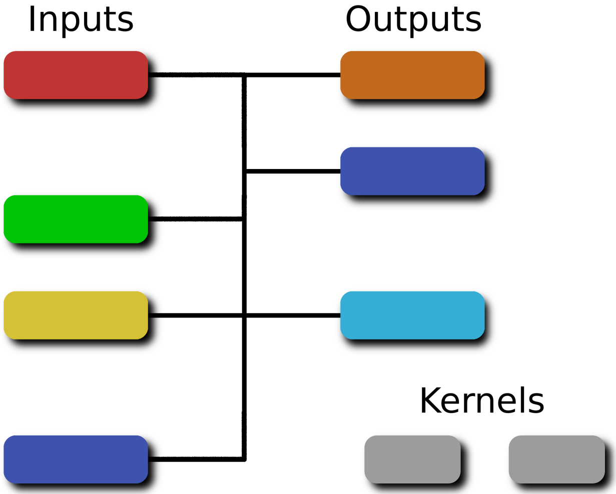
Kernels in Mimblewimble

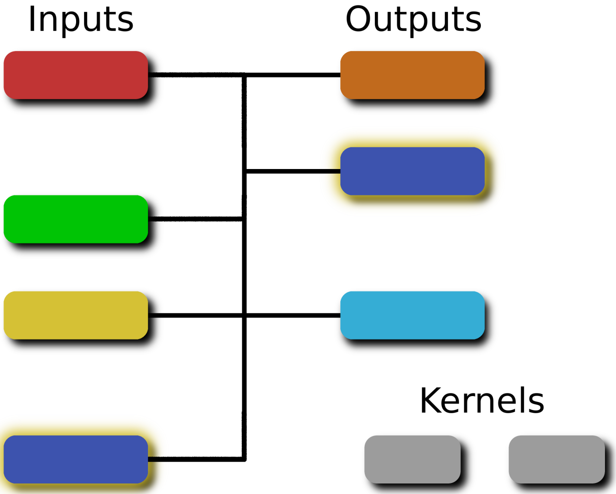
- Bitcoin (and Ethereum, etc.) uses a scripting language to describe smart contracts and enforce their execution.
- These scripts must be downloaded, parsed, validated by all full nodes on the network. Can't be compressed or aggregated.
- The details of the script are visible forever, compromising privacy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

Kernels in Mimblewimble

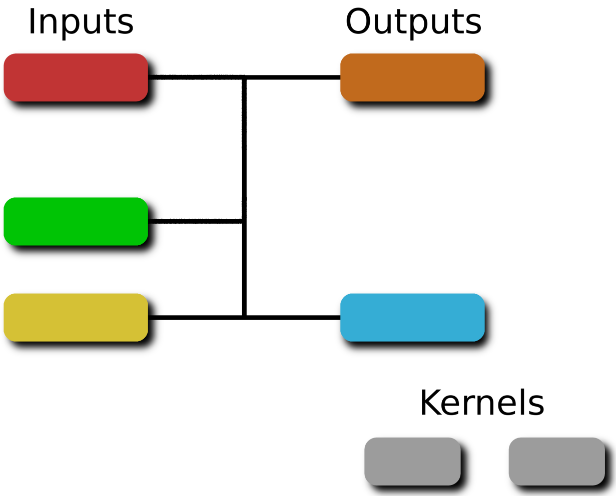


Kernels in Mimblewimble

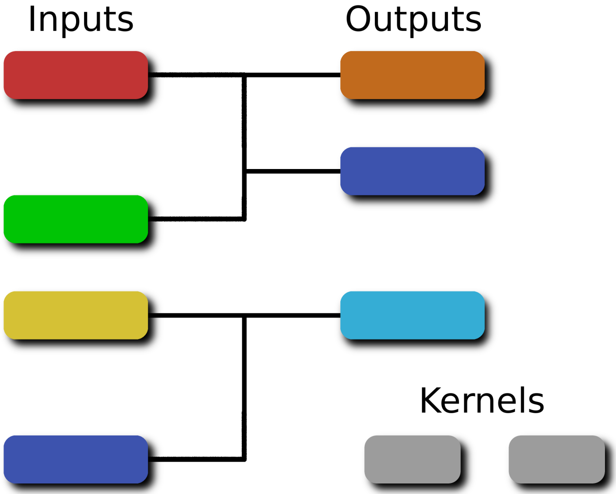



Inputs
Outputs
Kernels

Kernels in Mimblewimble



Kernels in Mimblewimble



Adaptor Signatures

- In a Schnorr multisignature, parties first exchange "public nonces" then exchange "partial signatures".
- Partial signatures are objects that, when added to other partial signatures, produce total signatures. This property is publicly verifiable.
- Given a secret t and commitment $T = tG$, you can offset partial signatures by t. The offset signature and T together are an adaptor signature.
- With an adaptor signature, if you learn the partial signature you'll also learn t, and vice-versa.

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 a commitment T to a secret value t.
- Bob replaces one of the signatures (s, R) with $(s + t, R)$ and publishes it, to take his coins. Alice sees this, learns t, then does the same thing on the other chain to take her coins.

Features of Adaptor Signatures

- By attaching auxiliary proofs to T to ensure t is some necessary data for a separate protocol, arbitrary steps of arbitrary protocols can be made equivalent to signature production.
- In particular, by using the same T in multiple adaptor signatures it is possible to make arbitrary sets of signatures atomic with other arbitrary sets, enabling multi-hop payment channels.
- You can re-blind commitments between hops while retaining the atomicity, for improved privacy.

Features of Adaptor Signatures

- After a signature hits the chain, anyone can make up a commitment T and compute a corresponding “adaptor signature” for it, so such schemes are *deniable*.
- Unlike hash-preimages, the secret t is revealed only to a party in possession of an adaptor signature, who can efficiently prove knowledge of it. This gives a *transferrable proof* that a protocol (e.g. a Lightning invoice) was completed correctly.
- Existing multisignature outputs can be used with adaptor signatures, no need to precommit to a specific protocol.

Limits of Adaptor Signatures

- Seem to really only be useful for 2-signer protocols. This includes 2-of-3 signatures.
- Depends on publication of complete signatures; seems incompatible with noninteractive signature aggregation or BLS.
- No clear way to do timelocks.

Interlude: Witness Encryption

- If one party has a secret they want to associate to an adaptor signature, they can encrypt this with t and provide a *zero-knowledge proof* that the encryption has t as a key and the secret is well-formed. ("Zero-Knowledge Contingent Payments", Maxwell 2011).

- But what if the party wants to encrypt to a secret they *don't* know? For example, a third party's signature (c.f. Discreet Log Contracts, Dryja 2017) or a future Bitcoin block?

- In 2013 we called general zk proofs "moon math". Today we use this term for *witness encryption* (Garg, Gentry, Sahai, Waters 2013).

Timelocks in Mimblewimble

- Absolute timelocks can be added to Mimblewimble by having kernels sign a minimum blockheight before which they may not be included in the blockchain.
- *Relative* timelocks are much harder, because kernel signatures are independent of transaction outputs, and transacton outputs are not even guaranteed to be visible to verifiers.
- Idea (Somsen, Friedenbach 2016, personal communication): let kernels reference other kernels, use them as "timelock references".
- Drawbacks: requires users plant extra kernels in the blockchain; requires validators maintain an index of kernels.

Open Problems

- Timelocks!!
- Scriptless scripts with BLS.
- Multiparty (3+ parties) scriptless scripts
- Standards & Interoperability

Thank You

Andrew Poelstra <grincon@wpsoftware.net>