Scriptless Scripts

Andrew Poelstra

grindelwald@wpsoftware.net

May 10, 2017
“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 is a blockchain design that supports only scriptless scripts, and derives its privacy and scaling properties from this.
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

- Schnorr signatures: signer has a keypair \((x, P)\).

- A signature is the public half of an “ephemeral keypair” \((k, R)\) along with a linear equation in \(x\) and \(k\). Equation depends on the hash of a message.

- Signature can be verified because the key-derivation map \(x \mapsto P\) is also linear.

- ECDSA signatures (used in Bitcoin) have the same basic shape but aren’t linear in \(x\) and \(k\), so they are less useful.
**Simplest (Sorta) Scriptless Script**

- OP\_RETURN outputs are used in Bitcoin to encode data for purpose of timestamping.

- Alternate: replace a public key $P$ with $P + \text{Hash}(P \parallel m)G$.

- Replacing the signer’s public key is called “pay to contract” and is used by Elements and Liquid to move coins onto a sidechain.

- Replacing the ephemeral key is called “sign to contract”. Used to attach a timestamp to an unrelated transaction with zero network overhead.
Schnorr multi-Signatures are Scriptless Scripts

- By adding Schnorr signature keys, a new key is obtained which can only be signed with the cooperation of all parties.

- Can be generalized to $m$-of-$n$ by all parties giving $m$-of-$n$ linear secret shares to all others so they can cooperatively replace missing parties.

- (Don’t try this at home: some extra precautions are needed to prevent adversarial choice of keys.)
Zero-Knowledge Contingent payments (Greg Maxwell): sending coins conditioned on the recipient providing the solution to some hard problem.

Recipient provides a hash $H$ and a zk-proof that the preimage is the encryption key to a valid solution. Sender puts coins in a script that allows claimage by revealing the preimage.

Use the signature hash $e$ in place of $H$ and now you have a scriptless script ZKCP: a single digital signature which cannot be created without the signer solving some arbitrary (but predetermined) problem for you.

Must be done as a multisig between sender and receiver so that the sender can enforce what $e$ is.
Simultaneous Scriptless Scripts

- Executing separate transactions in an atomic fashion is traditionally done with preimages: if two transactions require the preimage to the same hash, once one is executed, the preimage is exposed so that the other one can be too.

- Atomic Swaps (Tier Nolan) and Lightning channels (Poon/Dryja) use this construction.

- “Use the message-hash as the hash” doesn’t work here to scriptless-scriptify this because message hashes can’t be fixed before a signature is created. Worse, this would link the two transactions, violating the spirit of scriptless scripts.
Adaptor Signatures

- Instead use another ephemeral keypair \((t, T)\) and treat \(T\) as the “hash” of \(t\).

- When doing a multi-signature replace the old ephemeral key \(R\) with \(R + T\), and now the signature \(s\) must be replaced by \(s + t\) to be valid.

- Now the original \(s\) is an “adaptor signature”. Anyone with this can compute a valid signature from \(t\) or vice-versa. They can verify that it is an adaptor signature for \(T\), no trust needed.

- One can compute an adaptor signature without knowing \(t\), but they will then be unable to produce a real signature.
Atomic (Cross-chain) Swaps

- Parties Alice and Bob send coins on their respective chains to 2-of-2 outputs. Bob thinks of a keypair \((t, T)\) and gives \(T\) to Alice.

- Before Alice signs to give Bob his coins, she demands adaptor signatures with \(T\) from him for both his signatures: the one taking his coins and the one giving her coins.

- Now when Bob signs to take his coins, Alice learns \(t\) from one adaptor signature, which she can combine with the other adaptor signature to take her coins.
Basic Lightning

- Suppose Alice is paying David through Bob and Carol. She produces an onion-routed path

  \[\text{Alice} \rightarrow \text{Bob} \rightarrow \text{Carol} \rightarrow \text{David}\]

  and asks for public keys \(B\), \(C\) and \(D\) from each participant.

- She sends coins to a 2-of-2 between her and Bob. She asks Bob for an adaptor signature with \(B + C + D\) before signing to send him the coins.

- Similarly Bob sends coins to Carol, first demanding an adaptor signature with \(C + D\) from her. Carol sends to David, demanding an adaptor signature with \(D\).
Adaptor signatures work across blockchains, even if they use different EC groups, though this requires a bit more work.

After a signature hits the chain, anyone can make up a \((t, T)\) and compute a corresponding “adaptor signature” for it, so the scheme is deniable. It also does not link the signatures in any way.

Adaptor signatures are re-blindable, as we saw in the Lightning example. This is also deniable and unlinkable.
Mimblewimble is the ultimate scriptless script.

Every input and output has a key, and a transaction signature uses a multisignature of all these keys.

Transaction validity is now contained in a scriptless script; further, the signature has been used with other scriptless script constructions (atomic swaps, ZKCP, etc.) to add additional validity requirements with zero overhead or even visibility to the network.
Open Problems

- ECDSA support
- Locktimes and other extrospection
- Understanding the limits of scriptless scripts
Thank You

Andrew Poelstra <grindelwald@wpsoftware.net>