Scriptless Scripts

Andrew Poelstra

`grindelwald@wpsoftware.net`

March 4, 2017

"Scriptless Scripts"?

- Scriptless scripts: magicking digital signatures so that they can only be created by faithful execution of a smart contract.

“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

“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.

Why use Scriptless Scripts?

- Bitcoin (and Ethereum, etc.) uses a scripting language to describe smart contracts and enforce their execution

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.

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.
- Have little intrinsic structure to be compressed or aggregated

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.
- Have little intrinsic structure to be compressed or aggregated
- The details of the script are visible forever and compromise privacy and fungibility.

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.
- Have little intrinsic structure to be compressed or aggregated
- The details of the script are visible forever and compromise 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 secret key x, ephemeral secret key k. Publishes a public key xG.

Schnorr Signatures Support Scriptless Scripts

- Schnorr signatures: signer has a secret key x, ephemeral secret key k. Publishes a public key xG.
- A signature is the ephemeral public key kG as well as

$$s = k - ex$$

where $e = H(kG \| xG \| \mathrm{message})$.

Schnorr Signatures Support Scriptless Scripts

- Schnorr signatures: signer has a secret key x, ephemeral secret key k. Publishes a public key xG.
- A signature is the ephemeral public key kG as well as

  $$s = k - ex$$

  where $e = H(kG \| xG \| \text{message})$.
- Verified by checking

  $$sG = kG - exG$$

Schnorr Signatures Support Scriptless Scripts

- Schnorr signatures: signer has a secret key x, ephemeral secret key k. Publishes a public key xG.
- A signature is the ephemeral public key kG as well as

  $$s = k - ex$$

  where $e = H(kG \| xG \| \mathrm{message})$.
- Verified by checking

  $$sG = kG - exG$$

- ECDSA signatures (used in Bitcoin) have the same shape, but s lacks some structure and e commits to only the message.

Simplest (Sorta) Scriptless Script

- OP_RETURN outputs are used in Bitcoin to encode data for purpose of timestamping

Simplest (Sorta) Scriptless Script

- `OP_RETURN` outputs are used in Bitcoin to encode data for purpose of timestamping
- Instead, replace the public key (or emphemeral key) P with $P + \mathtt{Hash}(P \| m) G$.

Simplest (Sorta) Scriptless Script

- `OP_RETURN` outputs are used in Bitcoin to encode data for purpose of timestamping
- Instead, replace the public key (or emphemeral key) P with $P + \texttt{Hash}(P \| m)G$.
- Replacing the public key is called “pay to contract” and is used by Elements and Liquid to move coins onto a sidechain.

Simplest (Sorta) Scriptless Script

- `OP_RETURN` outputs are used in Bitcoin to encode data for purpose of timestamping
- Instead, replace the public key (or emphemeral key) P with $P + \texttt{Hash}(P \| m) G$.
- Replacing the public key is called "pay to contract" and is used by Elements and Liquid to move coins onto a sidechain.
- Replacing the emphemeral key is called "sign to contract" and can be used to append a message commitment in any ordinary transaction with zero network overhead.

Simplest (Sorta) Scriptless Script

- `OP_RETURN` outputs are used in Bitcoin to encode data for purpose of timestamping
- Instead, replace the public key (or emphemeral key) P with $P + \mathtt{Hash}(P \| m)G$.
- Replacing the public key is called "pay to contract" and is used by Elements and Liquid to move coins onto a sidechain.
- Replacing the emphemeral key is called "sign to contract" and can be used to append a message commitment in any ordinary transaction with zero network overhead.
- Works with Schnorr or ECDSA

multi-Signatures in Scriptless Script

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

multi-Signatures in Scriptless Script

- By adding Schnorr signature keys, a new key is obtained which can only be signed with with the cooperation of all parties.
- Can be generalized to m-of-n by all parties giving m-of-n shares to all others so they can cooperatively replace missing parties.

multi-Signatures in Scriptless Script

- By adding Schnorr signature keys, a new key is obtained which can only be signed with with the cooperation of all parties.
- Can be generalized to m-of-n by all parties giving m-of-n 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.)

multi-Signatures in Scriptless Script

- By adding Schnorr signature keys, a new key is obtained which can only be signed with with the cooperation of all parties.
- Can be generalized to *m*-of-*n* by all parties giving *m*-of-*n* 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.)
- Works with Schnorr only.

moSt exSpressive Scriptless Script

- Zero-Knowledge Contingent payments: sending coins conditioned on the recipient providing the solution to some hard problem.

moSt exSpressive Scriptless Script

- Zero-Knowledge Contingent payments: 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.

moSt exSpressive Scriptless Script

- Zero-Knowledge Contingent payments: 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.

moSt exSpressive Scriptless Script

- Zero-Knowledge Contingent payments: 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.

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 and Lightning channels use this construction.

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 and Lightning channels use this construction.
- The previous hash-preimage construction doesn't work because a signature hash can't be controlled like this, plus it would require nonce-reuse (breaking the signature security), plus it would link the two transactions, which violates the spirit of scriptless scipts.

Simultaneous Scriptless Scripts

- Instead what we do is consider the difference of two Schnorr signatures:

$$d = s - s' = k - k' + ex - e'x'$$

Simultaneous Scriptless Scripts

- Instead what we do is consider the difference of two Schnorr signatures:

$$d = s - s' = k - k' + ex - e'x'$$

- Given kG, $k'G$, e, e' this construction can be verified as

$$dG = kG - k'G + exG - e'x'G$$

Simultaneous Scriptless Scripts

- Instead what we do is consider the difference of two Schnorr signatures:

$$d = s - s' = k - k' + ex - e'x'$$

- Given kG, $k'G$, e, e' this construction can be verified as

$$dG = kG - k'G + exG - e'x'G$$

- Given d and either s or s', the other can be computed. So possession of d makes these two signatures atomic!

Simultaneous Scriptless Scripts

- Instead what we do is consider the difference of two Schnorr signatures:

$$d = s - s' = k - k' + ex - e'x'$$

- Given kG, $k'G$, e, e' this construction can be verified as

$$dG = kG - k'G + exG - e'x'G$$

- Given d and either s or s', the other can be computed. So possession of d makes these two signatures atomic!
- But since d is computable by anybody after s, s' are available, this scheme does nothing to link the two signatures or harm their security.

Sorceror's Scriptless Script

- MimbleWimble is the ultimate scriptless script.

Sorceror's Scriptless Script

- MimbleWimble is the ultimate scriptless script.
- Every input and output has a key (actually a Pedersen commitment, but the transaction balances exactly when these commitment behave like keys; this trick is Confidential Transactions).

Sorceror's Scriptless Script

- MimbleWimble is the ultimate scriptless script.
- Every input and output has a key (actually a Pedersen commitment, but the transaction balances exactly when these commitment behave like keys; this trick is Confidential Transactions).
- A transaction signature uses the multisignature key of all input and output keys (called a "kernel" in MimbleWimble parlance). It is irrelevant what gets signed, just that something is.

Sorceror's Scriptless Script

- MimbleWimble is the ultimate scriptless script.
- Every input and output has a key (actually a Pedersen commitment, but the transaction balances exactly when these commitment behave like keys; this trick is Confidential Transactions).
- A transaction signature uses the multisignature key of all input and output keys (called a "kernel" in MimbleWimble parlance). It is irrelevant what gets signed, just that something is.
- Transaction validity is now contained in a scriptless script; further, the signature has be used with other scriptless script constructions (atomic swaps, ZKCP, etc.) to add additional validity requirements with zero overhead.

Open Problems

- Generic scriptless scripts

Open Problems

- Generic scriptless scripts
- Locktimes or other extrospection

Thank You

Andrew Poelstra `<grindelwald@wpsoftware.net>`