Scriptless Scripts

Andrew Poelstra
Research Director, Blockstream
scriptless@wpsoftware.net

May 18, 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.”
• 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.
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.

- You can also make objects that, when added to arbitrary-but-precommitted values, produce total or partial signatures. These are adaptor signatures.

- With an adaptor signature, when you learn the partial signature you also learn the precommitted value, 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.
Suppose now that Alice is a mixing service who receives coins and sends coins, but does not want to be able to link sends to receives.

She can precommit to a *blind signature* sending coins to a user Bob, and give an adaptor signature to this blind signature.

Similar to the atomic swap, Bob sends Alice coins that can only be redeemed by her revealing the corresponding real signature.

Suppose now that Alice is a computational service who wants to sell the solution to some hard problem.

She can make a commitment $T$ to a value $t$, and provide a zero-knowledge proof that $t$ is the encryption key to a solution of the problem. (Also she provides the encrypted solution.)

She then sends an adaptor signature to $T$ to Bob, for a signature that would redeem coins jointly owned by her and Bob. Bob can now sign.
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.
New Developments

- Lightning with scriptless scripts (AJ Towns and others, lightning-dev 2018)

- ECDSA rather than Schnorr (Moreno-Sanchez, Kate 2018)
Thank You

Andrew Poelstra <wpsoftware.netless@wpsoftware.net>