Non-Interactive Bitcoin Covenants

Abstract

This proposal outlines a protocol for the creation of Bitcoin script conditions that restrict a UTXO to be spent to only a specific address.
This is done through the creation of a public key for which only a valid signature exists.

Design

To improve explainability, we will start by describing this proposal in the context of Schnorr signatures and will later explain how it can be ported to ECDSA:

Schnorr

Given a curve C, a generator point $G \in C$, a private key p and a nonce k, the public key $P \in C$ is usually computed as $P = pG$ and, given a message m a signature for it is created by computing $K = kG, s = k - hash(m, K) \cdot p$ and constructing (s, K, m), which along with P enables the verification of the signature.
The construction described allows anyone with knowledge of p to create a signature for P. But, instead of constructing a signature in this traditional way, this proposal uses the following algorithm:

1. Construct a transaction
2. Compute a definition of $s \in \mathbb{Z}_p$ and $K \in C$ in a deterministic way from the message, for example using $s = hash1(m), K = hash2(m)$ ($hash2$ should return a valid point in C)
3. Compute P by solving the equation $sG = K - hash(m, K) \cdot P$

ECDSA

The same scheme can be applied to ECDSA using any of several ECDSA recovery implementations, such as the `ecrecover` opcode that is part of the EVM.

THE Problem

There is a circular dependency between the transaction hash of the transaction that will initially move the funds to the address and the public key that is to be computed. This is caused by the fact that to compute the initial transaction we need to know the scriptPubKey where the funds should be sent, and that is constructed using the computed public key, yet to compute that key we need to know the hash of the transaction that should be signed (the transaction that would spend the funds from the address) and such a transaction is constructed using the txid of the transaction that initially sent the funds, thus:

- PubKey = f_1(hash(f_2(hash(InitialTransaction))))
- InitialTransaction = f_3(PubKey)

Where f_1, f_2 and f_3 are invertible functions.

Therefore, creating a PubKey or InitialTransaction that has the properties needed requires solving the following equation:
InitialTransaction = f_3(f_1(hash(f_2(hash(InitialTransaction)))))

Now, solving this equation would require generating hash pre-images, which should be impractical due to the computation expenses.

Due to this problem, the whole scheme is impossible to implement and would require the existence of something like `SIGHASH_NOINPUT` in order to be practical.

Fees

A basic problem with this design is the fact that only a single unique transaction can spend the UTXO, therefore the fees used on that transaction are locked and cannot be changed afterwards. Nevertheless it can be solved through:

- **Child Pays for Parent**
- **Several transactions with different fees**

Applications

This solution would enable the applications described in the `OP_CHECKOUTPUTSHASHVERIFY` BIP:

- Congestion Controlled Transactions
- Channel Factories
- Wallet Vaults
- More efficient CoinJoin

Furthermore, it also enables the following applications (these would also be possible with `OP_CHECKOUTPUTSHASHVERIFY`):

- Scripts with unbounded length (including arbitraryly sized multisigs)
- Arbitrary state machines (+ solves the problem of gas because it requires several transactions)
- Transaction-level MAST (a really coarse-grained version of MAST that would split code paths at the script/transaction level)

- Drivechains (although it would only let the sidechain send the locked coins to an address contained in a set of addresses that would have to be defined when creating the script, that is, it won't be able to spend the drivechain funds to arbitrary addresses, also the number of possible forward and backward steps would need to be artificially capped)

Deployment

This proposal can be applied directly and does not require any change on the Bitcoin protocol nor the node software.