

# 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  $K = kG, s = k - \text{hash}(m, K) \cdot p$  and constructing  $(s, K, m)$ , which along with  $P$  enables the verification of the signature.

The construction described allows the owner of Instead of constructing the signature in this way, this proposal uses the following algorithm:

Given a transaction input

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  $s = \text{hash1}(m), K = \text{hash2}(m)$  ( $\text{hash2}$  should return a valid point in  $C$ )
3. Compute  $P$  by solving the equation  $sG = K - \text{hash}(m, K) \cdot P$

### ECDSA

TODO

## 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 fund from the address) and such a transaction is constructed using the txid of the transaction that initially send the funds, thus:

- $\text{PubKey} = f\_1(\text{hash}(f\_2(\text{hash}(\text{InitialTransaction}))))$
- $\text{InitialTransaction} = f\_3(\text{PubKey})$

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:

$$\text{InitialTransaction} = f\_3(f\_1(\text{hash}(f\_2(\text{hash}(\text{InitialTransaction}))))))$$

Now this 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 OP\_NOINPUT in order to be practical.

## Fees

A basic problem with this design is the fact that . 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
- CoinJoin

Furthermore, it also enables the following applications (these would also be possible with OP\_CHECKOUTPUTSHASHVERIFY):

- Scripts with unbounded length (including arbitrarily sized multisigs)
- Arbitrary state machines (+solves the problem of gas bc 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.