```
BIP: 124
Layer: Applications
Title: Hierarchical Deterministic Script Templates
Author: Eric Lombrozo <eric@ciphrex.com>
        William Swanson <swansontec@gmail.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0124
Status: Rejected
Type: Informational
Created: 2015-11-20
License: PD
Post-History: http://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-November/011795.
```

## Abstract

This BIP defines a script template format that can be used by wallets to deterministically generate scripts with specific authorization policies using the key derivation mechanism defined in BIP32.

## Motivation

Currently existing wallets typically issue scripts in only a tiny handful of widely used formats. The most popular formats are pay-to-pubkey-hash and m-of-n pay-to-script-hash (BIP16). However, different wallets tend to use mutually incompatible derivation schemes to generate signing keys and construct scripts from them. Moreover, with the advent of hashlocked and timelocked contracts (BIP65, BIP112), it is necessary for different wallets to be able to cooperatively generate even more sophisticated scripts.

In addition, there's a lot of ongoing work in the development of multilayered protocols that use the blockchain as a settlement layer (i.e. the Lightning Network). These efforts require sufficiently generalized templates to allow for rapidly evolving script designs.

This BIP provides a generalized format for constructing a script template that guarantees that different wallets will all produce the same scripts for a given set of derivation paths according to BIP32.

## Specification

### Keys

An individual key is determined by a BIP32 derivation path and an index. For convenience, we introduce the following notation:

$\mathbf{A}_{\mathrm{k}}$ = (derivation path for A)/k

### Key Groups

Let $\mathbf{m}_\mathrm{i}$ denote distinct BIP32 derivation paths. We define a key group of n keys as a set of key derivation paths with a free index k:

$\{\mathbf{K}_\mathrm{k}\} = \{\ \mathbf{m}_1/\mathrm{k},\ \mathbf{m}_2/\mathrm{k},\ \mathbf{m}_3/\mathrm{k},\ ...,\ \mathbf{m}_\mathrm{n}/\mathrm{k}\ \}$

Key groups are useful for constructing scripts that are symmetric in a set of keys. Scripts are symmetric in a set of keys if the semantics of the script is unaffected by permutations of the keys. Key groups enforce a canonical form and can improve privacy.

### Sorting

We define a lexicographic sorting of the keys. (TODO: specification of sorting conventions - compressed pubkeys, encoding, etc...)

Define $\{\mathbf{K}_\mathrm{k}\}_\mathrm{j}$ to be the jth element of the sorted keys for derivation index k.

### Script Templates

We construct script templates by inserting placeholders for data into a script. To denote a placeholder, we use the following notation:

*Script*(**A**) = opcodes [**A**] opcodes

We extend this notation to an arbitrary number of placeholders:

*Script*(**X1**, **X2**, ..., **Xn**) = opcodes [**X1**] opcodes [**X2**] opcodes ... opcodes [**Xn**] opcodes

We introduce the following convenient notation for sorted key groups:

$[\{\mathbf{K}_\mathrm{k}\}] = [\{\mathbf{K}_\mathrm{k}\}_1]\ [\{\mathbf{K}_\mathrm{k}\}_2]\ ...\ [\{\mathbf{K}_\mathrm{k}\}_\mathrm{n}]$

### Operations on Keys

In some applications, we might want to insert the result of some operation performed on a key rather than the key itself into the script. For example, we might want to insert a Hash160 of key $\mathbf{A}_\mathrm{k}$. We can use the following notation:

[*Hash160*($\mathbf{A}_\mathrm{k}$)]

### Encoding

TODO

## Examples

### 2-of-3 Multisig

The script template is defined by:

*Script*(**X**) = 2 [**X**] 3 OP_CHECKMULTISIG

Letting $\mathbf{K}_k = \{ \mathbf{m}_1/k, \mathbf{m}_2/k, \mathbf{m}_3/k \}$, the $k$th script for this key group is denoted by *Script*($\{\mathbf{K}_k\}$).

#### 1-of-1 or 2-of-3

The script template is defined by:

*Script*(**A**, **B**) =
    OP_DUP [**A**] OP_CHECKSIG
    OP_NOTIF
        2 [**B**] 3 OP_CHECKMULTISIGVERIFY
    OP_NOTIF
    OP_ENDIF
    OP_TRUE

Let $\mathbf{M}_k = \mathbf{m}/k$ be a key of a superuser that can authorize all transactions and $\{\mathbf{K}_k\}$ be a key group of three users that can only authorize transactions if at least two of them agree.

The $k$th script is given by *Script*($\mathbf{M}_k$, $\{\mathbf{K}_k\}$).

#### Timelocked Contract

The output is payable to Alice immediately if she knows the private key for $\mathbf{A}_k$. Bob must know the private key for $\mathbf{B}_k$ and also wait for a timeout **t** before being able to spend the output.

The script template is defined by:

*Script*(**A**, **B**, **T**) =
    OP_IF
        OP_DUP OP_HASH160 [*Hash160*(**A**)] OP_EQUALVERIFY OP_CHECKSIG
    OP_ELSE
        [**T**] OP_CHECKLOCKTIMEVERIFY OP_DROP
        OP_DUP OP_HASH160 [*Hash160*(**B**)] OP_EQUALVERIFY OP_CHECKSIG
    OP_ENDIF

The $k$th script with timeout **t** is given by *Script*($\mathbf{A}_k$, $\mathbf{B}_k$, **t**).

## References


- BIP16 - Pay to Script Hash
- BIP32 - Hierarchical Deterministic Wallets
- BIP65 - OP_CHECKLOCKTIMEVERIFY
- BIP112 - CHECKSEQUENCEVERIFY
- Lightning Network Whitepaper

## Copyright