```
BIP: 389
Layer: Applications
Title: Multipath Descriptor Key Expressions
Author: Andrew Chow <andrew@achow101.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0389
Status: Draft
Type: Informational
Created: 2022-07-26
License: BSD-2-Clause
```

Abstract

This document specifies a modification to Key Expressions of Descriptors that are described in BIP 380. This modification allows Key Expressions to indicate BIP 32 derivation path steps that can have multiple values.

Copyright

Motivation

Descriptors can describe the scripts that are used in a wallet, but wallets often require at least two descriptors for all of the scripts that they watch for. Wallets typically have one descriptor for producing receiving addresses, and the other for change addresses. These descriptors are often extremely similar - they produce the same types of scripts, derive keys from the same master key, and use derivation paths that are almost identical. The only differences are in the derivation path where one of the steps will be different between the descriptors. Thus it is useful to have a notation to represent both descriptors as a single descriptor where one of the derivation steps is a pair of values.

Specification

For extended keys and their derivations paths in a Key Expression, BIP 380 states:

- `xpub` encoded extended public key or `xprv` encoded extended private key (as defined in BIP 32)
  - Followed by zero or more `/NUM` or `/NUMh` path elements indicating BIP 32 derivation steps to be taken after the given extended key.
  - Optionally followed by a single `/*` or `/*h` final step to denote all direct unhardened or hardened children.

This is modifed to state:

- `xpub` encoded extended public key or `xprv` encoded extended private key (as defined in BIP 32)
  - Followed by zero or more `/NUM` (may be followed by `h`, `H`, or `'` to indicate a hardened step) path elements indicating BIP 32 derivation steps to be taken after the given extended key.
  - Followed by zero or one `/<NUM;NUM` (each `NUM` may be followed by `h`, `H`, or `'` to indicate a hardened step) path element indicating a tuple of BIP 32 derivation steps to be taken after the given extended key.
    - Followed by zero or more `;NUM` (may be followed by `h`, `H`, or `'` to indicate a hardened step) additional tuple values of BIP 32 derivation steps
    - Followed by a single `>/`
  - Followed by zero or more `/NUM` (may be followed by `h`, `H`, or `'` to indicate a hardened step) path elements indicating BIP 32 derivation steps to be taken after the given extended key.
  - Optionally followed by a single `/*` (may be followed by `h`, `H`, or `'` to indicate a hardened step) final step to denote all direct unhardened or hardened children.

When a `/<NUM;NUM;...;NUM>` is encountered, parsers should account for a presence of multiple descriptors where the first descriptor uses the first `NUM`, and a second descriptor uses the second `NUM`, and so on, until each `NUM` is accounted for in the production of public keys, scripts, and addresses, as well as descriptor import and export operations. Descriptors that contain multiple Key Expressions that each have a `/<NUM;NUM;...;NUM>` must have tuples of exactly the same length so that they are derived in lockstep in the same way that `/*` paths in multiple Key expressions are handled.

The common use case for this is to represent descriptors for producing receiving and change addresses. When interpreting for this use case, wallets should use the first descriptor for producing receiving addresses, and the second descriptor for producing change addresses. For this use case, the element will commonly be the value `/<0;1>`

Note that only one `/<NUM;NUM;...;NUM>` specifier is allowed in a Key Expression.

Test Vectors

Valid multipath descriptors followed by the descriptors they expand into as sub-bullets

- `pk(xpub68NZiKmJWnxxS6aaHmn81bvJeTESw724CRDs6HbuccFQN9Ku14VQrADWgqbhhTHBaohPX4CjNLf9fq9M`
  - `pk(xpub68NZiKmJWnxxS6aaHmn81bvJeTESw724CRDs6HbuccFQN9Ku14VQrADWgqbhhTHBaohPX4CjNLf9`
  - `pk(xpub68NZiKmJWnxxS6aaHmn81bvJeTESw724CRDs6HbuccFQN9Ku14VQrADWgqbhhTHBaohPX4CjNLf9`
- `pkh(xprv9s21ZrQH143K31xYSDQpPDxsXRTUcvj2iNHm5NUtrGiGG5e2DtALGdso3pGz6ssrdK4PFmM8NSpSBHN`
  - `pkh(xprv9s21ZrQH143K31xYSDQpPDxsXRTUcvj2iNHm5NUtrGiGG5e2DtALGdso3pGz6ssrdK4PFmM8NSp`
  - `pkh(xprv9s21ZrQH143K31xYSDQpPDxsXRTUcvj2iNHm5NUtrGiGG5e2DtALGdso3pGz6ssrdK4PFmM8NSp`
- `wpkh([ffffffff/13h]xpub69H7F5d8KSRgmmdJg2KhpAK8SR3DjMwAdkxj3ZuxV27CprR9LgpeyGmXUbC6wb7E`

  - wpkh([ffffffff/13h]xpub69H7F5d8KSRgmmdJg2KhpAK8SR3DjMwAdkxj3ZuxV27CprR9LgpeyGmXUbC6
  - wpkh([ffffffff/13h]xpub69H7F5d8KSRgmmdJg2KhpAK8SR3DjMwAdkxj3ZuxV27CprR9LgpeyGmXUbC6
- multi(2,xpub6ERApfZwUNrhLCkDtcHTcxd75RbzS1ed54G1LkBUHQVHQKqhMkhgbmJbZRkrgZw4koxb5JaHWkY
  - multi(2,xpub6ERApfZwUNrhLCkDtcHTcxd75RbzS1ed54G1LkBUHQVHQKqhMkhgbmJbZRkrgZw4koxb5Ja
  - multi(2,xpub6ERApfZwUNrhLCkDtcHTcxd75RbzS1ed54G1LkBUHQVHQKqhMkhgbmJbZRkrgZw4koxb5Ja
- pkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCGu8ED9W6oDMSgv6Zz8idoc4a
  - pkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCGu8ED9W6oDMSgv6Zz8id
  - pkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCGu8ED9W6oDMSgv6Zz8id
  - pkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCGu8ED9W6oDMSgv6Zz8id
- sh(multi(2,xpub6ERApfZwUNrhLCkDtcHTcxd75RbzS1ed54G1LkBUHQVHQKqhMkhgbmJbZRkrgZw4koxb5JaH
  - sh(multi(2,xpub6ERApfZwUNrhLCkDtcHTcxd75RbzS1ed54G1LkBUHQVHQKqhMkhgbmJbZRkrgZw4koxb
  - sh(multi(2,xpub6ERApfZwUNrhLCkDtcHTcxd75RbzS1ed54G1LkBUHQVHQKqhMkhgbmJbZRkrgZw4koxb
  - sh(multi(2,xpub6ERApfZwUNrhLCkDtcHTcxd75RbzS1ed54G1LkBUHQVHQKqhMkhgbmJbZRkrgZw4koxb

Invalid descriptors

- Multiple multipath specifiers: pkh(xprv9s21ZrQH143K31xYSDQpPDxsXRTUcvj2iNHm5NUtrGiGG5e2DtALGds
- Multipath specifier in origin: pkh([deadbeef/<0;1>]xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMskt
- Multipath specifiers of mismatched lengths: tr(xpub661MyMwAqRbcF3yVrV2KyYetLMYA5mCbv4BhrKwUrFE9
- Multipath specifiers of mismatched lengths: sh(multi(2,xprvA1RpRA33e1JQ7ifknakTFpgNXPmW2YvmhqLQ
- Empty multipath specifier: wpkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCG
- Missing multipath start: wpkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCGu8
- Missing multipath end: wpkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syAmRUapSCGu8E
- Missing index in multipath specifier: wpkh(xpub661MyMwAqRbcFW31YEwpkMuc5THy2PSt5bDMsktWQcFF8syA

Backwards Compatibility

This is an addition to the Key Expressions defined in BIP 380. Key Expressions using the format described in BIP 380 are compatible with this modification and parsers that implement this will still be able to parse such descriptors. However as this is an addition to Key Expressions, older parsers will not be able to understand such descriptors.

This modification to Key Expressions uses two new characters: `<` and `;`. These are part of the descriptor character set and so are covered by the checksum algorithm. As these are previously unused characters, old parsers will not accidentally mistake them for indicating something else.

This proposal is in contrast to similar proposals such as BIP 88 which allow for multiple derivation indexes in a single element. This limitation exists in order to reduce the number of descriptors that are expanded, avoid confusion about how to expand the descriptor, and avoid having expanded descriptors that users are not expecting.

Reference Implementation

https://github.com/bitcoin/bitcoin/pull/22838