```
BIP: 88
Layer: Applications
Title: Hierarchical Deterministic Path Templates
Author: Dmitry Petukhov <dp@simplexum.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0088
Status: Proposed
Type: Informational
Created: 2020-06-23
License: BSD-2-Clause
```

Abstract

This document describes a format for the representation of the templates that specify the constraints that can be imposed on BIP32 derivation paths.

The constraints specified by the templates allow to easily discern 'valid' paths, that match the constraints, and 'invalid' paths, that exceed the constraints.

Copyright

Motivation

BIP32 derivation path format is universal, and a number of schemes for derivation were proposed in BIP43 and other documents, such as BIPs 44,45,49,84. The flexibility of the format also allowed industry participants to implement custom derivation shemes that fit particular purposes, but not necessarily useful in general.

Even when existing BIPs for derivation schemes are used, their usage is not uniform across the different wallets, in part because software vendors might have different considerations and priorities when making decisions about derivation paths. This creates friction for users, which might face problems when they try to access their coins using the wallet that derives addresses differently than the one they used before.

Known solutions

The problem is common enough to warrant the creation of a dedicated website (walletsrecovery.org) that tracks paths used by different wallets.

At the time of writing, this website has used their own format to succintly describe multiple derivation paths. As far as author knows, it was the only publicitly used format to describe path templates before introduction of this BIP. The format was not specified anywhere beside the main page of the website. It used

`|` to denote alternative derivation indexes (example: `m/|44'|49'|84'/0'/0'`) or whole alternative paths (`m/44'/0'/0'|m/44'/1'/0'`).

It was not declared as a template format to use for processing by software, and seems to be an ad-hoc format only intended for illustration. In contrast to this ad-hoc format, the format described in this BIP is intended for unambigouos parsing by software, and to be easily read by humans at the same time. Humans can visually detect the 'templated' parts of the path more easily than the use of `|` in the template could allow. Wider range of paths can be defined in a single template more succintly and unambiguously.

Intended use and advantages

Wallet software authors can use the proposed format to describe the derivation paths that their software uses. This can improve user experience when switching to different wallet software, restoring access to old wallets, etc.

Unrestricted derivation path usage might be unsafe in certain contexts. In particular, when "change" outputs of a transaction are sent to the addresses derived via paths unknown to the sender, the sender might lose access to the whole change amount.

A simplistic approach of hard-coding the checks for well-known paths into software and firmware leads to reduced interoperability. Vendors cannot choose custom paths that are appropriate for their particular, non-general-purpose applications, and are forced to shoehorn their solutions into using well-known paths, or convince other vendors to support their custom paths. This approach scales poorly.

A flexible approach proposed in this document is to define a standard notation for "BIP32 path templates" that succintly describes the constraints to impose on the derivation path.

Wide support for these path templates will increase interoperability and flexibility of solutions, and will allow vendors and individual developers to easily define their own custom restrictions. This way, they will be able to deal with the risks of accidental or malicious use of unrestricted derivation paths in a more flexible and precise manner.

Well-known path templates can be pre-configured by default on devices and applications, but users can have an option to turn off the templates that are not relevant to their uses.

Having a standardized format for custom path templates will enable a common approach to be developed in the enforcement of application-specific path restrictions in devices and applications. One example of such an approach might be for devices to allow application-specific profiles with path templates and possibly other custom parameters. Care must be taken to prevent the accidental installation of malicious or incorrect profiles, though.

Specification

The format for the template was choosen to make it easy to read, convenient and visually unambigous.

Template starts with optional prefix `m/`, and then one or more sections delimited by the slash character (`/`).

Implementations MAY limit the maximum number of sections.

Each section consists of *index template*, optionally followed by the hardened marker: either an apostrophe (`'`) or letter `h`.

Index template can be:

- An integer value from 0 to 2147483647 ("Unit index template")
- A single `*` character, which denotes any value from 0 to 2147483647 ("Wildcard index template")
- The `{` character, followed by a number of *index ranges* delimited by commas (`,`), followed by `}` character ("Ranged index template")

Implementations MAY limit the maximum number of index ranges within the Ranged index template.

If an index template is immediately followed by hardened marker, this means that all values specified in this index template is to be increased by 2147483648 for the purposes of matching.

Index range can be:

- An integer value from 0 to 2147483647 ("Unit range")
- An integer value from 0 to 2147483647, followed by the `-` character, followed by another integer value from 0 to 2147483647 ("Non-unit range")

For Non-unit range, value on the left side of the `-` character is the range_start, and the value on the right side of the `-` character is the range_end.

For Unit range, we say that range_start is equal to range_end, even though there is no start/end in the Unit range.

Unit index template contains a single index range, which is the Unit range

Wildcard index template contains a single index range, and we say that its range_start is set to 0 and its range_end is set to 2147483647

Constraints:

1. To avoid ambiguity, whitespace MUST NOT appear within the path template.
2. Commas within the Ranged index template MUST only appear in between index ranges.
3. To avoid ambiguity, an index range that matches a single value MUST be specified as Unit range.

4. To avoid ambiguity, an index range `0-2147483647` is not allowed, and MUST be specified as Wildcard index template instead
5. For Non-unit range, range_end MUST be larger than range_start.
6. If there is more than one index range within the Ranged index template, range_start of the second and any subsequent range MUST be larger than the range_end of the preceeding range.
7. To avoid ambiguity, all representations of integer values larger than 0 MUST NOT start with character `0` (no leading zeroes allowed).
8. If hardened marker appears within any section in the path template, all preceding sections MUST also specify hardened matching.
9. To avoid ambiguity, if a hardened marker appears within any section in the path template, all preceding sections MUST also use the same hardened marker (either `h` or `'`).
10. To avoid ambiguity, trailing slashes (for example, `1/2/`) and duplicate slashes (for example, `0//1`) MUST NOT appear in the template.

It may be desireable to have fully unambiguous encoding, where for each valid path template string, there is no other valid template string that matches the exact same set of paths. This would enable someone to compare templates for equality through a simple string equality check, without any parsing.

To achieve this, two extra rules are needed:

- Within Ranged index template, subsequent range MUST NOT start with the value that is equal to the end of the previous range plus one. Thus, `{1,2,3-5}` is not allowed, and should be specified as `{1-5}` instead. This rule might make templates less convenient for frequent edits, though.

- Only one type of hardened marker should be allowed (either `h` or `'`).

Instead of requiring the second extra rule, implementations can simply replace one type of marker with another in the template strings before comparing them.

Full and partial templates

If the template starts with `m/`, that means that this is the "full" template, that matches the whole path.

If the template does not start with `m/`, that means that this is a "partial" template, and it can be used to match a part of the path, in the contexts where this might be appropriate (for example, when constraints for the suffix of the path might be dynamic, while constraints for the prefix of the path are fixed).

Full template can be combined with partial template, where partial template extends full template, resulting in new, longer full template.

Partial template can be combined with another partial template, resulting in new, longer partial template.

Full template can not be combined with another full template.

Implementations MUST support parsing full templates and matching paths against full templates.

Implementations MAY support parsing partial templates and matching portions of the paths against partial templates, as well as combining the templates.

Parsing result

The result of successful parsing of a valid path template can be represented by a list of sections, where each section is a list of index ranges, where index range is a tuple of (range_start, range_end). The length of the list of sections is also referred to as the "length of the template".

Matching

The matching is to be performed against a list of integer values that represent a BIP32 path (or a portion of BIP32 path, for partial templates). The length of this list is referred to as the "length of the path".

Non-hardened indexes in this list should be represented by values from 0 to 2147483647.

Hardened indexes in this list should be represented by values from 2147483648 to 4294967295.

The matching algorithm:

```
   1. If the length of the path differs from the length of the template, fail
2. For each value V at position N in the path:
If for all index ranges within the section at position N in the template,
value V is either less than range_start, or greater than range_end, fail
3. Otherwise, succeed
```

Formal specification

The finite state machine (FSM) for the parser of the described template format, and the matching formula are specified in TLA+ specification language at https://github.com/dgpv/bip32_template_parse_tplaplus_spec

The specification can be used with TLC checker and accompanying script to generate test data for the implementations.

Implementations

While the formal specification specifies an FSM, which would be convenient for implementation without access to rich string handling facilities, when such facilities are available, the implementation might use the whole-string deconstruction approach where the templates are first split into sections, then sections are split into index templates, and then each index template are parsed individually.

A FSM-based approach can be made close to the formal specification, though, and the test data generated with TLC checker would give much better coverage for a FSM based implementation. If the template string contains several errors, an implementation that uses deconstruction approach might detect some of these errors earlier than FSM-based implementation, and vise versa.

At the moment, three implementations exist:

- FSM implementation in C: https://github.com/dgpv/bip32_template_c_implementation
- FSM implementation in Python (micropython compatible): https://github.com/dgpv/bip32_template_python_implementation
- non-FSM implementation in python: BIP32PathTemplate class in bitcointx.core.key module of python-bitcointx library (https://github.com/Simplexum/python-bitcointx)

Compatibility

The full path template that only contains Unit index templates represents a fully valid BIP32 path.

There's no other path template standards that is known to the author currently.

There is a discussion on path templating for bitcoin script descriptors at https://github.com/bitcoin/bitcoin/issues/17190, which proposes the format `xpub...{0,1}/*`, of which the `{0,1}/*` part would correspond to the partial path template in the format of this BIP.

Examples

`m/{44,49,84}'/0'/0'/{0-1}/{0-50000}` specifies a full template that matches both external and internal chains of BIP44, BIP49 and BIP84 paths, with a constraint that the address index cannot be larger than 50000

Its representation after parsing can be (using Python syntax, ignoring full/partial distinction):

```
   [[(2147483692, 2147483692), (2147483697, 2147483697), (2147483732, 2147483732)),
[(2147483648, 2147483648)],
[(2147483648, 2147483648)],
[(0, 1)],
[(0, 50000)]]
```

`{0-2,33,123}/*` specifies a partial template that matches non-hardened values 0, 1, 2, 33, 123 as first index, and any non-hardened value at second index

Its representation after parsing can be:

```
   [[(0, 2), (33, 33), (123, 123)], [(0, 2147483647)]]
```

`*h/0` specifies a partial template that matches any hardened index followed by non-hardened index 0

Its representation after parsing can be:

```
[[(2147483648, 4294967295)], [(0, 0)]]
```

Acknowledgements

Special thanks to Peter D. Gray, Dr. Maxim Orlovsky, Robert Spigler and others for their feedback on the specification, and to Janine (github:@Enegnei) for the help in preparing the draft.