Sawtooth Lake 0.8
Setting Key Naming Strategy

DRAFT

Overview

In order to facilitate selecting a subset of on-chain settings by key, a key-to-state-address conversion strategy is needed to satisfy the following constraint: a portion of the key corresponds to a portion of the state address. For example, a user only wishes to select keys under the “poet.*” grouping, but ignore the settings under “sawtooth.*”.

Design

Addresses are made up of 70 hexadecimal digits, with the first 6 making up a transaction family namespace, and the remaining 64 make up the address of the data within the namespace. With on-chain settings, the namespace is `000000`. The remainder of the data is made up from a series of hashes of key parts.

In order to allow users to to query for groups of settings, without having to query the whole of the the settings, there needs to be a correspondence between the key parts and the address.

The remaining 64 bytes of a setting address can be broken up into 4 parts, each corresponding to a part of the setting key. Each part of the key is hashed with sha256 and the first 16 characters of the hash are used. For keys with less than 4 parts, the remaining hashes are produced using empty strings. For keys with more than 4 parts, the last part will be the remaining subkey (that is, 3 individual parts, and the 4 made up of the remain subkey).

Breaking it into 4 parts limits the granularity of subqueries, for deeply nested settings, but a consistent position for the keys is necessary for subgroup queries to be predictable, without the application developer requiring knowledge of how deeply nested the settings are.

Examples

Taking the example key of `a.b`, the breakdown is as follows:

“a”	ca978112ca1bbdca
“b”	3e23e8160039594a
“”	e3b0c44298fc1c14
“”	e3b0c44298fc1c14

With the namespace, the resulting address is:
`000000ca978112ca1bbdca3e23e8160039594ae3b0c44298fc1c14e3b0c44298fc1c14`

Taking the example key of `a.b.c.d.e.f`

“a”	ca978112ca1bbdca
“b”	3e23e8160039594a
“c”	`2e7d2c03a9507ae2`
“d.e.f”	`7d1b4186ed404d5e`

With the namespace, the resulting address is:
`000000ca978112ca1bbdca3e23e8160039594a2e7d2c03a9507ae27d1b4186ed404d5e`

The subgroup of keys under `a.*` can be queried using the prefix:
`000000ca978112ca1bbdca`
Both of the above setting keys and the values of the setting entries would be included in the returned leaf-nodes.

Considerations

The choice of 4 parts has been selected for a reasonable amount of depth in a nested settings tree. This number of parts can be increased or decreased, so long as it's an even divisor of 64. In the case of deeply nested keys, the production of address require more and more sha256, which does increase the computation of addresses.

It does limit meaningful subqueries to the first 3 parts of the keys, but the likelihood of an application developer nesting that far *and* being interested in querying subparts of the settings (without being interested in a single key) would be low, as well as not a recommended way of organizing settings.