One of the features of the sharding/stateless client model is that a transaction needs to fully specify the set of accounts (or more generally, the portion of the state) that it can read and write to, in order to preserve the assurance that a transaction can be executed with (possibly an updated version of) the same witness data that it was sent with.

The earlier version of sharding did this by simply providing an account list; because the storage tree was replaced by a storage bytearray, every account was O(1)-sized, and so the witness size for a transaction could be bounded.

Now, we are making two modifications to sharding v1:

- Going back to a storage tree model, with the justification that this reduces merkle proof size, and the original argument that a single bytearray can be more easily fetched from disk is less relevant in the stateless client model
- A two-layer model where the account structure including every account's storage tree is embedded inside a single trie

This implies that the witness for a single account is no longer necessarily O(1) sized, and in fact we want to be able to give a maximally fine-grained ability to specify specific subsets of storage as a witness.

This gives rise to the following scheme:

[[address, prefix1, prefix2...], [address, prefix1, prefix2...], ...]

For every address given, the nonce/balance/code become part of the access list automatically, and then the remaining prefixes are prefixes of storage (eg. an empty prefix would represent the entire storage tree, a prefix 0x15dcf9 would represent all storage keys starting with that value, and a full 32-byte prefix would represent a single storage key).

We can then translate this into "prefix list form", referring to key prefixes for the actual underlying trie:

def to_prefix_list_form(access_list): o = [] for obj in access_list: addr, storage_prefixes = obj[0], obj[1:] o.append(sha3(addr) + b'\x00') o.append(sha3(addr) + b'\x01') o.append(sha3(addr) + b'\x02') for prefix in storage_prefixes: o.append(sha3(addr) + b'\x03' + prefix) return o

This prefix list form would then be relatively easy for client implementations to work with to calculate required witnesses.

Note that our original representation provides a high level of expressiveness, allowing subtrees of storage to be added to the access list, and also is efficient - there is rarely a need to repeat any prefix (in contract, the trie prefix list form is NOT efficient as account addresses would get repreated many times).

HLLs (solidity, viper, etc) could take advantage of this and segregate mappings and variables in the storage by assigning each variable and mapping a different byte prefix, thereby allowing transactions that, for example, claim something like "I am allowed to access the entire dict foo

, and the key bar.baz

, and the variable x

but not other things.