Today the public data tree stores one field element per leaf. While the storageWrite

oracle call accepts multiple elements, under the hood it gets converted to multiple update requests to sequential storage slots. Updates can be batched together, but reading data that's multiple fields long requires one merkle membership proof for each element.

One option to optimize this is to "merge" the merkle membership proofs for reading. Since each element in a new array (or struct) is inserted one after the other, their corresponding nodes in the indexed merkle tree should land together, so most of the merkle path would be shared. This means a merkle membership proof can provide the longest common subpath to the root (ie the path of the smallest subtree that contains all elements), and then just the paths from the elements to the subtree root.

However, it's unclear how easy it is to implement this optimization. Also, in worst case this optimization does not offer any gains, since the elements could land right in the middle of the tree in which case the smallest shared subtree is the entire tree.

An easier solution is to allow storing data of arbitrary length in each leaf of the tree. Instead of storing the actual value, we store a hash of the value in each leaf. Public update requests could push the hash of the data, and emit its preimage as an event-like. All in all, we replicate the same structure as in private state: the private state tree stores commitments to data, and the actual data (notes) gets emitted as events.

Thoughts? Does this increased complexity pay off for the savings in proving time from the fewer hashings needed when reading or writing data that comprises multiple fields?