TLDR

: We present a “multihashing” trick to make Ethereum 2.0 consensus objects natively compatible with multiple hash functions.

Credits

: This idea came up in a discussion with@barrywhitehat and @vbuterin.

Construction

Given “simple” hash functions h_1, ..., h_n

define the “multihash function” H = [h_1, ..., h_n]

. Build consensus objects (e.g. beacon blocks or the beacon state) using H

instead of any individual h_i

.

Candidate hash functions h_i

include Keccak256, Friday, MiMC, SHA256, Blake2b.

Discussion

Hash functions are a key building block for blockchains, and enshrining one hash function over another has significant tradeoffs. Multihashing is a way to (largely) escape those tradeoffs through “hash subjectivity”. That is, infrastructure interacting with core L1 objects (include other L1 objects) can chose the hash function(s) to work with and hence make appropriate tradeoffs.

More concretely, below are potential benefits of multihashing:

- Compatibility

: Using Keccak256 as a constituent hash function makes Ethereum 1.0 and 2.0 compatible. This unlocks Ethereum 1.0 deposits to Ethereum 2.0, as well as Ethereum 1.0 execution engines that use the Ethereum 2.0 data layer.

- Flexibility

: By including exotic hash functions in the multihash (e.g. Friday or MiMC) we unlock the possibility for SNARK/STARK-based gadgets (at all layers). This includes data availability proofs, witness compression, STARK-based quantum-secure infrastructure, SNARK-based privacy infrastructure, etc.

- Interoperability

: In the spirit of Polkadot and Cosmos, multihashing increases interoperability with other blockchains. For example, adding SHA256 as a constituent hash function increases interoperability with major blockchains such as Bitcoin.

- Security

: If a particular hash function (e.g. Keccak256) is found unsafe the Ethereum 2.0 consensus layer maintains security through the other constituent hash functions.

- Efficiency

: EVM 2.0 dApps can save gas when interacting with consensus objects by using a cheap hash function (e.g. Blake2b) .

Downsides of multihashing:

- Overhead

: This includes larger consensus objects (32*n-byte multihashes instead of 32-byte hashes) and increased CPU-time to compute the multihash.

- Complexity

: Multiple hash functions are part of the consensus logic. (Note that this may be unavoidable for S[T/N]ARK-friendly hash functions.)

- Lack of default

: A single hash function is a Schelling point for the ecosystem to standardise around.