TLDR

: We present a data custody scheme based on bitwise XOR. It is a simplification of the Merkleisation-based custody scheme and drastically improves its challenge communication complexity. That is, the worst-case number of challenge-response rounds as well as the worst-case data overhead is significantly reduced.

Construction

Let s

be a 32-byte secret and D

an n

-bit piece of data. We define helper functions:

- Let chunkify(D, k)

be the ordered list of k

equally-sized chunks of D

.

- Let M

be the “mix” of D

with s

, i.e. the concatenation of hash(xor(c, s))

for every chunk c

in chunkify(D, len(D)/32)

.

- Let bitwise_xor(D)

to be the XOR of every bit of D

.

A prover wanting to prove custody of D

prepares the mix M

, commits to the custody bit bitwise_xor(M)

and the secret s

, and later reveals s

.

A challenger that disagrees on the custody bit issues a k

-bit challenge where the i

th bit is bitwise_xor(chunkify(M, k)[i])

. The prover must then disagree with the challenger on some i_0

th bit with corresponding chunk M_0 = chunkify(M, k)[i_0]

. The prover then responds with i_0

alongside a k

-bit response where the i

th bit is bitwise_xor(chunkify(M_0, k)[i])

. The challenger must then disagree with the prover on some i_1

th bit with corresponding chunk M_1 = chunkify(M_0, k)[i_1]

. The game continues until either the prover or challenger responds with a k

-bit chunk of M

alongside a Merkle proof.

Illustration

In the context of Ethereum 2.0 sharding (phase 1), assume that SHARD_BLOCK_SIZE = 2**15

bytes and that SLOTS_PER_EPOCH = 2**6

slots. A validator wanting to prove custody of shard blocks over an epoch prepares a mix M

with size n = 2**24

bits (2MB). We illustrate the challenge game with two different values of k

.

When k = 2**12

, the mix M

is split into k

chunks, each of size k

bits. If the challenger disagrees with bitwise_xor(M)

it shares bitwise_xor

for every chunk of M

. The validator must disagree on bitwise_xor

for some i_0

th chunk. It then responds with i_0

, the corresponding chunk M_0

, and a Merkle proof for M_0

. The round complexity is 1 (two half rounds), and the total communication overhead is 1378 bytes:

- k

bits for the challenge (512 bytes)

- log(k)

bits for i_0

(2 bytes)

- k

bits for M_0

(512 bytes)

- log(k) - 1

hashes for the Merkle path of M_0

(352 bytes)

When k = 2**8

, the mix M

is split into k**2

chunks, each of size k

bits. If the challenger disagrees with bitwise_xor(M)

it shares bitwise_xor(chunkify(M, k)[i])

for every 0 <= i < k

. The validator must disagree on some M_0 = chunkify(M, k)[i_0]

, and shares i_0

and bitwise_xor(chunkify(M_0, k)[i])

for every 0 <= i < k

. The challenger must disagree on some M_1 = chunkify(M_0, k)[i_1]

which happens to be a chunk of M

. It then responds with i_1

, M_1

, and a Merkle proof for M_1

. The round complexity is 1.5 (three half rounds), and the total communication overhead is 578 bytes:

- k

bits for the initial challenge (32 bytes)

- log(k)

bits for i_0

(1 byte)

- k

bits for M_0

(32 bytes)

- log(k)

bits for i_1

(1 byte)

- k

bits for M_1

(32 bytes)

- 2*log(k) - 1

hashes for the Merkle path of M_1

(480 bytes)