Problem statement

: for use cases like [Optimizing sparse Merkle trees](), create a hash function $H(l, r) = x$

where l

, r

and x

are 32 byte values that is (i) collision-resistant and (iii) trivial to compute if $l = 0$

or $r = 0$

. This ensures that sparse trees with 2^{256}

virtual nodes only require $log(N)$

“real” hashes to be computed to verify a branch or make an update to an average N

-node tree, all while preserving the very simple and mathematically clean interface of a sparse Merkle tree being a simple binary tree where almost all of the leaves are zero.

Algorithm

1. If $l \ne 0$

and $r \ne 0$

, return $2^{240} + sha256(l, r)\ mod\ 2^{240}$

(ie. zero out the first two bytes of the hash)

1. If $l = r = 0$

return 0

1. If $l \ge 2^{255}$

or $r \ge 2^{255}$

or $l < 2^{240}$

or $r < 2^{240}$

, return $2^{240} + sha256(l, r)\ mod\ 2^{240}$

1. Otherwise let x

be the nonzero input and b

be 1 if r

is nonzero else 0. Return $2 * x + b$

Collision resistance argument

- If $h = 0$

, then it can only have come from case (2) as preimage resistance of $f(x) = sha256(x)\ mod\ 2^{240}$

implies that finding l

and r

that hash to zero is infeasible so cases (1) and (3) are ruled out, and case 4 is ruled out because either value being nonzero makes $2 * x + b$

nonzero.

- Outputs $1 \le h <2^{240}$

are outright impossible as none of the four cases can produce them

- Outputs $2^{240} \le h < 2^{241}$

can only have come from cases 1 or 3 (as for them to come from case 4, an input $x \in [2^{239}, 2^{240})$

would be required, which cannot happen as case 3 catches that possibility). Collision resistance of $f(x) = sha256(x)\ mod\ 2^{240}$

implies that there is at most one discoverable solution.

- Outputs $2^{241} \le h$

can only have come from case 4. $floor(\frac{h}{2})$

identifies the only possible value for the nonzero input, and $h\ mod\ 2$

identifies which of the inputs was nonzero.

Properties

At the cost of a 16-bit reduction in preimage resistance and 8-bit reduction in collision resistance, we get the property that hashing a sparse tree with one element with depth d

only requires about $1 + \frac{d}{16}$

“real” hashes.