<u>Index code</u>

msg	*EC*1	*EC*2

msg of length $k = 2^m - 1$ bits.
EC = bitwise XOR over indices of active (1) bits in *msg* (indices start at 1)
We transmit EC twice! That is, two copies of EC

example:

msg = 0110110 (k=2^3-1)
indices 1 2 3 4 5 6 7

2= 010 ⊕
3= 011 ⊕
5= 101 ⊕
6= <u>110</u>
EC= 010

transmission = 0110110<u>010010</u>

<u>Note that</u>: *m* bits are needed to represent each copy of EC
Explanation:

decimal x is represented by $\lfloor \log_2(x) \rfloor + 1$ bits.

$|\boldsymbol{EC}| = \boldsymbol{O}(\boldsymbol{m})$ since $\lfloor \log_2(2^m - 1) \rfloor + 1 = m$

So we add logarithmically many bits for each copy of EC: n = $2^m - 1 + O(m)$
(worse than O(1) for parity, better then O(k) for repetition):

<u>Question:</u> d=?

d>=3: It is not possible to have 2 (legal) codewords of distance < 3:

- If two msgs differ in 1 bit, their EC1 must be different (The ECs will differ exactly in the positions where the binary representation of the different bit contains 1) → overall: at least 3 differences between the two codewords.
  The same holds for EC2.
- If two msgs differ in 2 bits, both their EC1 and EC2 will differ in at least one bit. (because two different indices cannot cancel each other in the EC computation) → overall: at least 4 differences between the two codewords.

d<=3: We give an example of two (legal) codewords of distance 3:
0000000<u>000000</u> and 0001000<u>100100</u>. (any index which is power of 2 would work)

→d=3

Can detect 2 errors, fix 1.

An improvement: add parity bit at the end.

msg	*EC*1	*EC*2	*p*

Claim: The distance of the code is **d=4**.

Proof outline: Before, when d=3, the closest codewords were 3 bits apart. Because they differed in an odd number of bits, their parity bits are different, and so the total distance between them (including the parity) is now 4. In addition, the distance between all other codeword pairs cannot decrease following the addition of a new bit, and therefore the code distance increased by 1.

In general, if a certain code has distance d, the addition of a parity bit will create a new code with distance d', such that:

- If the original distance d was odd, then the new distance will be d' = d+1
- If the original distance d was even, then the new distance will be d' = d (distance will not change).

Since **d=4**, we can detect 3 errors, fix 1 error.