Index code

msg	EC

msg of length $k = 2^m - 1$ bits.
EC = bitwise XOR over indices of active (1) bits in *msg* (indices start at 1)

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>010</u>

<u>Question</u> |EC|= O(…)?

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: n = $2^m - 1 + O(m)$
(worse than O(1) for parity, better then O(k) for repetition):

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

d>=2: It is not possible to have 2 (legal) codewords of distance 1:
If two msgs differ in 1 bit, their EC must be different (The ECs will differ exactly in the positions where the binary representation of the different bit contains 1) → overall: at least 2 differences between the two codewords.

d<=2: We give an example of two (legal) codewords of distance 2:
0000000<u>000</u> and 0001000<u>100</u>. (any index which is power of 2 would work)

→d=2

Can detect 1 error, fix 0.

First improvement: transmit *EC* twice.

msg	*EC*1	*EC*2

The new distance is **d=3**.
Proof is very similar to previous one. We need to show also that two changes in msg cannot cancel each other, and must lead to at least one change in both EC1 and EC2.

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). The same holds for EC2. → overall: at least 3 differences between the two codewords.
- 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 2:
0000000<u>000000</u> and 0001000<u>100100</u>. (any index which is power of 2 would work)

→d=3

Can detect 2 error, fix 1.

Decoding algorithm (<u>assumes at most 1 error has occurred</u>):

<u>decode (trans = msg+EC1+EC2):</u>

1. compute *EC* ' from msg
2. if *EC* ' = *EC*1 or *EC* ' = *EC*2 #if both, then 0 errors
3. return msg # no error in *data*
4. else: # *EC*1 = *EC*2, single error in *msg*
5. i = *EC* ' ⊕ *EC*1 # or ⊕EC2, doesn't matter. Index of error.
5. i = int(i,2) #to decimal
6. return $msg[:i-1]$+ $\overline{msg[\imath-1]}$ + $msg[i:]$ #bit i flipped

<u>Example</u>:

$\boldsymbol{encoding}$: 0110110 → 0110110<u>010010</u>

$\boldsymbol{error}$: 0110110<u>010010</u> → 0110$\overline{0}$10<u>010010</u>

$\boldsymbol{decoding}$: 0110010<u>010010</u>

$$EC' = 2 \oplus 3 \oplus 6 = 010 \oplus 011 \oplus 110 = 111 \neq 010$$

$$conclusion: error\ at\ bit\ 111 \oplus 010 = 101\ (= 5)$$

$\boldsymbol{return}$: 0110$\overline{1}$10

In case 2 errors have occurred, our algorithm may sometimes return the wrong msg.

Example: We use the same transmission from above, but with bits 2,5 flipped due to errors: 0010010010010

EC': 3⊕6 = 011⊕110 = 101

We would conclude a single error at $101 \oplus 010 = 111$ (which is 7 in decimal) and return 001001**1** (now with 3 errors).

Second 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).

Claim: After the addition of the parity bit, the distance between every two codewords is always even.

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