Hail Zeus (Crypto, 300p)

> Asking Hermes to gath'r intel on the foe, he recover'd the blueprint of their transmitt'r bef're getting captur'd. It's been ov'r a month of radio silence. but wait... all hail Zeus, as they restart'd communication when he strok'd them with a lightning:

###ENG PL

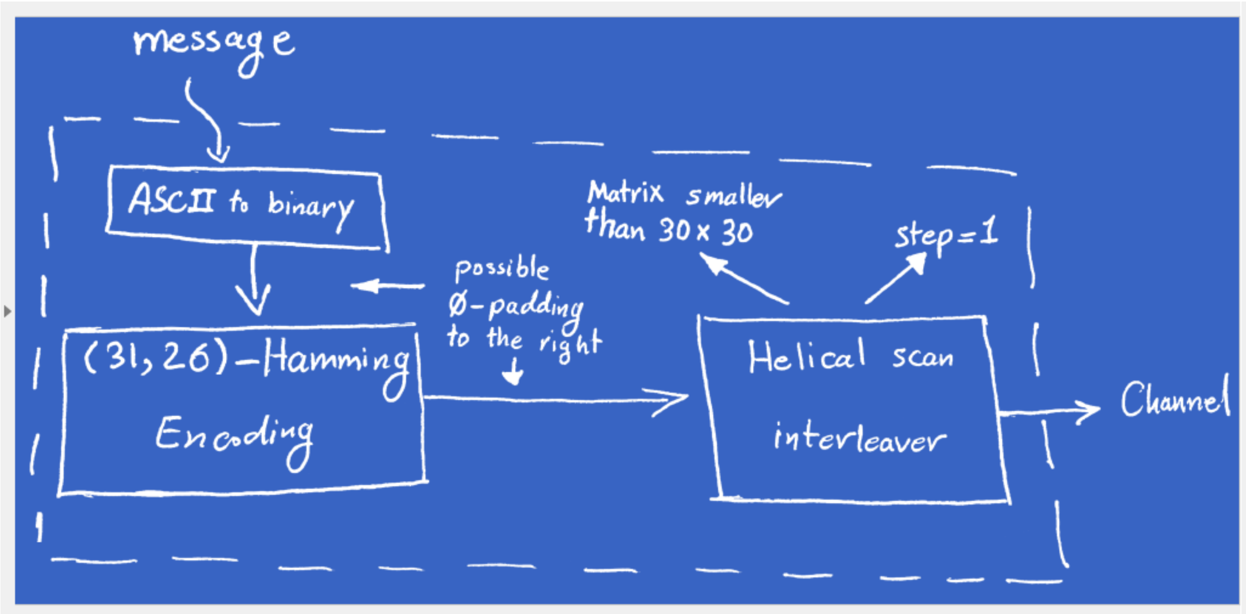
Although this challenge wa worth "only" 300 points, only one team managed to solve it (our team of course, that's why i can write this writeup right now).

We are given long bit string representing "enemy communication", and we have to decode it:

```
 1111101010101001010101101010001111010100000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000001001011010111111111000001111001010101110010100101110111001011111100
0011010001111110001011001101100110011110010010111010110111101101101010100000101101
10000110101011110100001010100000110001011111000000111111001000010001111110110100101
0100101001010111000110000000111001101010000000111111111101000011111000101100101010110
1000010011000101110101010100000011110000110101010101010000111100001101010000111101
000011010010010101111100101011111001111010001001101010000000001001100001011000000111
1110000110011000110110011010110011110010010001101111000000101111000100100010100101
0001111010110111110010010011011110010001000000101110101000100000000011000110011100111
0111001011000100010010011001000111110000111100000011001110110100000010100001100011
000101010110010000001000100001110010010110000111110010000101010111101001011101000010
111100110110110111110100100101100011010000001000000000010000111010001010110111100
10010011000010011001101111000000101100011110000001111100100111000111011000101010001
00111101001111001011100011110010000011000000011010111011000101000001101010000111110
10010011100111011010111101101111111000010001000010110001110100011110000100000001001
0011000110010000000001000101110001001011011111100100111010000110101010111000010011
0001001010001011010101101010100101100011001100101101000100101000110110000100101011
0111001001010000001011001101101000011001010011110100110100000110111000010100010110
0100111100100100101000101110011110110000101001001111100010110101110111010100010110
0011010011101001001010010110100100011101101000111000111000000110110011001001000010
01100011011000000101001000110001110100010111101010010011010100000001110101000011100
0101100010001111010100001010110000010111010011010000011010000011000100011001001110
1001001010010100111000000111000011111110000110101101110001011010100001101000011000
0001101001011010000001011000110111001101001001100010011110010011100000110110100110
10001111111010111110000010010000110100100011110010110011001010001011110000001101010
0011000100011000000110001001001011011111101001110111011010110100101100010101001001
1000100101100101001100111111000000000100001010000000011100101001000111101000000011001
000001000101001111011101010001111100111101100010110100111100000111011001110100010
0000100010101101000011001110010011010010010100010111010000110100000111000000001010
1101011000010110100001100100101010000011110101011111010001010111111011000011100001
1100000111100000111101101111000000111000000100100000111011001110010010010001010011
```

```
000011010000010001001001000001001011010011010011101000010111010010111010001100001
011000011010010010110101011000000010010100010111001011101101111010101111000010101
000011111000001000110111110101101101011001010011111000101101001011001000110100101
011001001111101110010000010101000011000000110101000111001000101010001111110011110
100010010100010011000010110000111010101101000101011101000001000010100000001101010
110111000100100000100100000011011010111001010110011001110100111110100001001101100
011001011010111001110110110010110101000100100110100101111110001010110010000011110
010111010011000110110100001101000000110010011110100110110010011000100111110100001
001001110010011111000100001000011100010111100001110101000111010001011101001001000
000000000001101100110010000100101000001100000111010000011000010010000010010100110
000000001000000000000000110111000000000000111000000000001100101000000001111011000000
001010001000000000110010000000000001001100000000100010000000000011010010000000001011
010000000010000...
```

We are also given blueprint describing transmission method:



As you can see, ASCII message is encoded with hamming code, and then interleaved with helical scan matrix.

Before we start reversing this transmission, we have to learn something about hamming codes and helical scan matrices.

Hamming codes are family of error correcting, linear codes. They have advantage over simple checksums, because they can be used to repair simple single-bit flips, not only to detect them.

In practice using hamming codes can be reduced to multiplying by appropiate matrices - generato generator matrix G and parity check matrix H.

Every size of hamming code need different matrix, so we have to find (or compute, it's not that hard) appropiate matrix for our needs. In case of hamming(31, 26) matrix we need looks like this:

```
mat_g = [
    [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0],
    [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
    [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1],
```

```
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1],
]

mat_h = [
    [1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0],
    [0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0],
    [1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0],
    [0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0],
    [0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1],
]
```

We also need some way to multiply matrices:

```
def mult(a, b):
    rows_a = len(a)
    cols_a = len(a[0])
    rows_b = len(b)
    cols_b = len(b[0])

    if cols_a != rows_b:
        print "cannot multiply the two matrices. Incorrect dimensions:", cols_a, rows_b
        return

    c = [[0 for row in range(cols_b)] for col in range(rows_a)]

    for i in range(rows_a):
        for j in range(cols_b):
            c[i][j] = sum(a[i][k] * b[k][j] for k in range(cols_a)) & 1
    return c


def transpose(mat):
    return [[mat[y][x] for y in range(len(mat))] for x in range(len(mat[0]))]

assert transpose([[1, 2, 3], [4, 5, 6]]) == [[1, 4], [2, 5], [3, 6]]
```

Decoding hamming codes is very easy - it's enough to ignore error correcting/error checking informations (if message is not damaged of course). We can detect if message is damaged by multiplying it with matrix H - if message is not corrupted we should get zero matrix in result.

Second part of transmission is helical scan interleaving. Data is saved in matrix, and read in different order. For example:

```
                1   2  3  4
                5   6  7  8
```

1 2 3 4 ... 14 15 16 -> 9 10 11 12 -> 1 6 11 16 5 10 15 4 9 14 3 8 13 2 7 12 13 14 15 16

(We read first element in first row, and proceed diagonally. Then we read first element in second row and proceed diagonally, etc).

We need some helper methods, to convert between matrices from raw data strem (both ways):

```
def make_matrix(w, h, data):
    return [[data[i*w+j] for j in range(w)] for i in range(h)]

assert make_matrix(2, 3, [1, 2, 3, 4, 5, 6]) == [[1, 2], [3, 4], [5, 6]]


def unmake_matrix(w, h, data):
    return [data[i/w][i%w] for i in range(w*h)]

assert unmake_matrix(2, 3, [[1, 2], [3, 4], [5, 6]]) == [1, 2, 3, 4, 5, 6]
```

And encoding/decoding:

```python
def chunks(data, n, pad_obj=0):
    pad = list(data) + [pad_obj] * (n-1)
    return [pad[i*n:(i+1)*n] for i in range(len(pad)/n)]

assert chunks([1, 2, 3, 4, 5], 3) == [[1, 2, 3], [4, 5, 0]]


def helical_interleave_part(w, h, dat):
    mat = make_matrix(w, h, dat)
    conv = [[mat[(y+x) % h][x] for x in range(w)] for y in range(h)]
    return unmake_matrix(w, h, conv)

assert helical_interleave_part(2, 3, [1, 2, 3, 4, 5, 6]) == [1, 4, 3, 6, 5, 2]


def helical_interleave(w, h, dat):
    return sum((helical_interleave_part(w, h, part) for part in chunks(dat, w*h)), [])

assert helical_interleave(2, 3, [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]) == [1, 4, 3, 6, 5, 2, 7, 10, 9, 12, 11, 8]


def helical_deinterleave_part(w, h, dat):
    mat = make_matrix(w, h, dat)
    conv = [[mat[(y-x) % h][x] for x in range(w)] for y in range(h)]
    return unmake_matrix(w, h, conv)

assert helical_deinterleave_part(2, 3, [1, 4, 3, 6, 5, 2]) == [1, 2, 3, 4, 5, 6]


def helical_deinterleave(w, h, dat):
    return sum((helical_deinterleave_part(w, h, part) for part in chunks(dat, w*h)), [])
```

We have implemented everything we need to solve this challenge. We are given all necessary information to actually decode transmission - except size of helical scan matrix. Fortunatelly, we know that matrix is smaller than 30x30, so we can just bruteforce width and height. We will know that we guessed right by checking amount of hamming code errors. Random data will rarely have correct checksum (1/32 chance), and correctly deinterleaved data should have most checksums correct.

So moving on to implementation phase:

```python
data = open('data.txt').read().strip()
data = [int(c) for c in data]

result = []

for w in range(1, 30):
    print w, ':',
    for h in range(1, 30):
        print h,
        fail = 0

        helix = helical_deinterleave(w, h, data)
        cs = chunks(helix, 31)
        for c in cs:
            hamming_check = mult(mat_h, transpose([c]))
            hamming_check = transpose(hamming_check)
            if not all(n == 0 for n in hamming_check[0]):
                fail += 1

        result.append((fail, w, h))
    print

def safe(s):
    return ''.join(c if 32 <= ord(c) <= 127 else '.' for c in s)

result = sorted(result)

fail, w, h = result[0]
print 'best result:', fail, w, h
```

Our code tells us that matrix is 24 elements wide and 16 elements high. We can just decode data now, right?

```
helix = helical_deinterleave(w, h, data)
helix = decode_helix_brute(mat_g, mat_h, helix)
dat = chunks(helix, 8)
decr = [int(''.join(str(c) for c in chunk), 2) for chunk in dat]
decr_hex = ''.join(chr(c) for c in decr).encode('hex')
decr_bin = bin(int(decr_hex, 16))[2:]
```

Unfortunately, there is one more thing we have to do - we don't know from which bit we should start decoding (transmission is not byte-aligned). But that's non-issue, because we can just bruteforce all 8 possiblities:

```
for i in range(8):
    data = repr(''.join([chr(int(''.join(chunk), 2)) for chunk in chunks(decr_bin[i:], 8, '0')]))
    if 'SharifCTF' in data:
        print data
```

```
y weighs you down and torments you with regret. Drawing rooms, gossip, balls, SharifCTF{4412e6635c6eafaad08574d77ab4c
and triviality --- these are the enchanted circle I cannot escape from. I am now going to the war, the greatest war t
and I know nothing and am fit for nothing. I am very al\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

Challenge solved, 300 points (+ 100 bonus) well earned.

Working code is inside hamming.py file.

###PL version

Zacznę może od tego, że mimo ze zadanie ma "tylko" 300 punktów, to zostało rozwiązane tylko przez jedną drużynę na świecie - p4 (naszą drużynę tzn). Jako że nikt poza nami nie opisze tego zadania (bo nikt inny go nie zrobił), czuję się zobowiązany do opisania naszego rozwiązania.

Ale do rzeczy. Dostajemy bardzo długi ciąg bitów, będący "komunikacją" o której mowa w treści zadania:

```
 1111101010101001010101101010001111010100000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000010010110101111111110000011110010101011100101001011101110010111100
00110100011111100010110011011001100111100100101110101101111011011010101000010110 1
10000110101011110100001010100000110001011111000001111110010000100011111101101001 0
01001010010101110001100000011100110101000000111111111101000011111000101100101011 0
10000100110001011101010101000000111100001101010101010100001111000011010100011110 1
00001101001001010111110010101111100111101000100110101000000000100110000101100000 11
11100001100110001101100110101100111100100100011011110000010111100010010001010010 1
00011110101101111001001001101111001000100000101110101000100000000110001100111001 1
01110010110001000100100110010001111110000111100000011001110110100000101000011000 11
00010101011001000000100010000111001001011000011111001000010101011110100101110100 10
11110011011011011111010010010110001101000000100000000000100001110100010101101111 00
10010011000010011001101111000000101100011110000001111100100111000111011000101010 001
00111110100111100101110001111001000001100000001101011101100010100000110101000011 10
10010011100111011010111101101111111000010001000010110001110100011110000100000100 1
00110001100100000000010001011100010010110111111001001110100001101010101110000100 11
00010010100010110101011010101001011000110011001011010001001010001101100001001010 1
01110010010100000010110011011010000110010100111101001101000001101110000101000101 10
01001111001001001010001011100111101100001010010011111000101101011101110101000101 0
00110100111010010010100101101001000111011010001110001110000001101100110010010001 0
01100011011000000101001000110001110100010111101010010011010100000011101010000111 00
01011000100011110101000010101100000010111010011010000011010000011000100011001001 10
10010010100101001110000001110000111111100001101011011100010110101000011010000110 00
00011010010110100000010110001101110011010010011000100111100100111000000110110100 110
10001111111010111100000010010000110100100011110010110011001010001011110000011010 10
00110001000110000001100010010010110111111010011101110110110100101100010101001001 1
10001001011001010011001111111000000000100001010000000011100101001000111101000000 1100
00000100010100111101110101000111110011110110001011010011110000011101100111010001 0
00001000101011010000110011100100110100100101000101110100001101000000111000000001 010
11010110000101101000011001001010100000111101010111110100010101111110110000111000 01
11000001111000000111101101111000000011100000010010000001110110011100100100100010 10011
00001101000000100010010010000010010110100110100111010000101110100101110100011000 01
01100001101001001011010101100000000100101000101110010111011011110101011110000101 01
00001111100000010001101111101011011010110010100111110001011010010110010001101001 01
01100100111110111001000000101010000110000000110101000111001000101010001111110011 110
```