Task: Stream Cipher implementation:

Input a key K - 36 bytes (288 bits) and message M - of N-bytes $8 * N$ bits to yield output C - also N bytes.

(The code should be able to handle N upto $(2^{16} - 1)$. The key is 36 bytes (288 bits), to be read from a binary file Key. The file should have at least 36 bytes and you read the first 36 bytes (even if file is longer).

The plaintext is also to be read from binary file, Input (it could be say a mp3 file or a jpg file or a text file).

The output - ciphertext is to be written to a binary file Output. All the file input - output should be done in bytes. The bits in a byte are to be read left to right (just as a convention).

The names of Input Output and Key files should NOT be hardcoded but passed on the command line as follows:

$ python stream.py Input Output Key
or for C++, if the executable is named stream
$ stream Input Output Key

The decryption will be achieved by the same command:
$ python stream.py Output decoded_input Key
$ stream Output decoded_input Key

The streamcipher algorithm would generate a stream of bytes
$\{z[t]\}$
(depending upon the key) and if the message bytes are $m[0], m[1], \ldots, m[N-1]$**, then the bytes of the ciphertext** $c[0], c[1], \ldots, c[N-1]$ **are defined as follows:**

$$c[j] = m[j] \oplus z[j], \quad 0 \leq j < N$$

The streamcipher algorithm depends upon 8-LFSRs specified and a permutation P **of the set** $\{0, 1, 2, \ldots, 255\}$**.**

These are specified below:

For $i = 0, 1, \ldots, 7$, the i^{th} LFSR has degree $n[i]$ and primitive polynomial $p[i]$ specified below by:

$$
\begin{aligned}
n[0] &= 41,\ p[0](x) = x^{41} + x^3 + 1,\\
n[1] &= 29,\ p[1](x) = x^{29} + x^2 + 1,\\
n[2] &= 39,\ p[2](x) = x^{39} + x^4 + 1,\\
n[3] &= 31,\ p[3](x) = x^{31} + x^3 + 1,\\
n[4] &= 35,\ p[4](x) = x^{35} + x^2 + 1,\\
n[5] &= 28,\ p[5](x) = x^{28} + x^3 + 1,\\
n[6] &= 49,\ p[6](x) = x^{49} + x^9 + 1,\\
n[7] &= 36,\ p[7](x) = x^{36} + x^{11} + 1
\end{aligned}
$$

Let P be a permutation of $\{0, 1, \ldots, 255\}$ given by (chosen randomly)

$P =$ {99,217,3,113,189,127,235,224,120,142,79,78,24,45,218,177,198,141,203,51,251,181,
163,112,4,67,91,216,240,164,124,146,28,172,81,75,61,36,212,93,144,11,26,237,219,
94,44,66,35,122,173,135,100,73,200,76,145,117,211,126,92,132,202,232,82,154,210,129,
201,8,41,214,152,161,182,220,98,1,248,187,239,231,77,85,68,165,25,138,155,27,30,206,
43,131,209,125,195,140,153,225,247,37,207,255,80,84,188,33,22,7,167,105,16,72,108,
139,13,34,168,242,191,160,205,107,147,169,47,49,171,222,101,159,70,204,223,233,38,
236,65,234,254,151,118,19,57,229,158,116,20,150,40,197,137,143,10,157,14,175,252,123,
136,134,64,246,42,21,62,111,249,149,109,90,60,128,186,199,88,87,48,253,102,148,74,133,
58,213,23,54,115,121,228,31,174,55,0,170,185,166,190,178,18,59,179,110,196,245,238,
162,156,63,53,130,71,15,184,12,97,89,5,183,17,104,106,103,250,243,9,50,56,114,227,86,
29,180,208,244,96,241,52,32,83,221,95,192,2,193,6,230,226,46,176,119,215,194,39,69}

The parameters $n[i]$, $0 \leq i < 8$ and the primitive polynomials $p[i]$, $0 \leq i < 8$ as well as the permutation P are to be hardcoded in the program file.

Details of algorithm that would generate $\{z[t] : 0 \le t < N\}$.

The initial bits of the 8-LFSR's:

$$\{x[i][j] : 0 \le j < n[i], \ 0 \le i < 8\}$$

are specified as follows:

From the keyfile - the code should read first 288 bits and set $K : k[0], k[1], \ldots, k[287]$ (36 bytes) and let $s[0] = 0$, for $1 \le i < 8$ let $s[i] = s[i-1] + n[i-1]$ and

$$x[i][j] = k[s[i] + j], \ 0 \le j < n[i], \ \ 0 \le i < 8.$$

If the key file has ABCDEF..., then the first 6 characters are ABCDEF and in ASCII/ UTF-8 code will be 65,66,67,68,69,70 - in bits it would read: (for easy counting, I am alternating colours after 8-bits)
010000010100001001000011010001000100010101000110

Thus, $x[0][0], x[0][1], \ldots, x[0][41]$ would be
010000010100001001000011010001000100010101

Denoting the coefficients of the polynomials $p[i]$, $0 \le i < 8$ as

$$p[i] = 1 + \sum_{j=1}^{n[i]} c[i][j] x^j$$

the i^{th} LFSR's defines a sequence of bits of arbitrary finite length N^* by

$$x[i][t] = \Big(\sum_{j=1}^{n[i]} c[i][j] \otimes x[i][t-j]\Big) \text{ modulo } 2, \ \ n[i] \le t < n[i] + N^*$$

Using the array (2dim)

$$\{x[i][t] : n[i] \leq t < n[i] + N + 4,\ 0 \leq i < 8\}$$

we define (2dim) arrays
$\{a[i][t], b[i][t], c[i][t]\}$ for $0 \leq t < N$, $0 \leq i < 8$ as follows:
Let $h[i] = (i + 3) \bmod 8$ and $g[i] = (i + 5) \bmod 8$ and

$$a[i][t] = x[i][n[i] + t]$$

$$b[i][t] = x[h[i]][n[h[i]] + t + 2]$$

$$c[i][t] = x[g[i]][n[g[i]] + t + 5]$$

Let $f(a, b, c) = (a \otimes c) \oplus (b \otimes (1 + \oplus c))$.
Let (1-dim) arrays u, v, w, z be defined by, for $0 \leq t < N$

$$u[t] = \sum_{i=0}^{7} 2^i f(a[i][t], b[i][t], c[i][t])$$

$$v[t] = \sum_{i=0}^{7} 2^i f(b[i][t], c[i][t], a[i][t])$$

$$w[t] = \sum_{i=0}^{7} 2^i f(c[i][t], a[i][t], b[i][t])$$

and

$$z[t] = P[u[t]] \oplus P[v[t]] \oplus P[w[t]]$$

This completes the description of the algorithm.