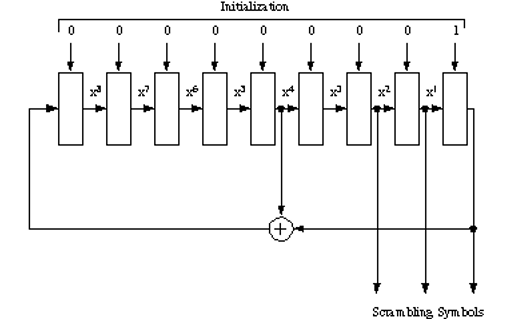
**Data Scrambling**:

Data symbols for the 8PSK symbol constellation shall be scrambled by modulo-8 addition with a scrambling sequence. The data symbols for the 16QAM, 32QAM, and 64QAM constellations shall be scrambled by using an exclusive or (XOR) operation. Sequentially, the data bits forming each symbol (4 for 16QAM, 5 for 32QAM, and 6 for 64QAM) shall be XOR’d with an equal number of bits from the scrambling sequence. In all cases, the scrambling sequence generator polynomial shall be **x9 +x4 +1** and the generator shall be initialized to 1 at the start of each data frame. A block diagram of the scrambling sequence generator for 8-PSK is shown in **Figure 1**.



**FIGURE 1. Scrambling sequence generator illustrating scrambling generator for 8PSK symbols.**

For 8-PSK symbols, the scrambling shall be carried out taking the modulo-8 sum of the numerical value of the binary triplet consisting of the last (rightmost) three bits in the shift register, and the symbol number (transcoded value). For example, if the last three bits in the scrambling sequence shift register were 010 which has a numerical value equal 2, and the symbol number before scrambling was 6, symbol 0 would be transmitted since: (6+2) Modulo 8 = 0. For 16QAM symbols, scrambling shall be carried out by XORing the 4-bit number consisting of the last (rightmost) four bits in the shift register with the symbol number. For example, if the last 4 bits in the scrambling sequence shift register were 0101 and the 16QAM symbol number before scrambling was 3 (i.e. 0011), symbol 6 (0110) would be transmitted. For 32QAM symbols, scrambling shall be carried out by XORing the 5-bit number formed by the last (rightmost) five bits in the shift register with the symbol number. For 64QAM symbols, scrambling shall be carried out by XORing the 6-bit number formed by the last (rightmost) six bits in the shift register with the symbol number.

After each data symbol is scrambled, the generator shall be iterated (shifted) the required number of times to produce all new bits for use in scrambling the next symbol (i.e., 3 iterations for 8PSK, 4 iterations for 16QAM, 5 iterations for 32QAM and 6 iterations for 64QAM). Since the generator is iterated after the bits are use, the first data symbol of every data frame shall, therefore, be scrambled by the appropriate number of bits from the initialization value of 00000001.

The length of the scrambling sequence is 511 bits. For a 256-symbol data block with 6 bits per symbol, this means that the scrambling sequence will be repeated just slightly more than 3 times, although in terms of symbols, there will be no repetition.

The MATLAB Function to do Scrambling is as below, however, it is required to modify the MATLAB code to run for performance i.e. less run-time and lesser memory requirements.

%%% Function: [scr\_dbit] = data\_scrambler(in\_dbit,enab,M)

%%% Description:

%%% Function to perform a data scrambling on data symbols. The

%%% scrambling sequence generator polynomial shall be x9 +x4 +1

%%% and the generator shall be initialized to 1 at the start of

%%% each data frames.

%%% Use modulo-8 operation for 8-PSK and XOR for 16-QAM & 64-QAM

%

%%% Inputs: -in\_dbit ==> Input data bits to scramble

%%% enab ==> To enable scrambler

%%% M ==> M-ary value; M = 2^k; where k= Number of

%%% bit group to form a symbol

%%% Output: -scr\_dbit ==> Output scramble data bits

%%% mod\_array ==> Modulated array for all the constellation

%%% points

%

%%% Author: NIL

%%% Version: 1.0

%%% Date: 1st Nov 2017

function [scr\_dbit] = data\_scrambler(in\_dbit,enab,M)

k = ceil(log2(M));

L = length(in\_dbit);

scr\_data = [1 zeros(1,8)]; %% Initialize the scrambler

scr\_data\_calc(:,1) = scr\_data; %% Store the very first scrambler out as init data

for ii=1:L-1

for kk=1:k

scr\_data = [scr\_data(2:end) xor(scr\_data(1),scr\_data(5))];

end

scr\_data\_calc(:,ii+1) = scr\_data;

scr\_data\_rev = scr\_data\_calc(1:4,:);

end;

for jj=1:L

in\_bits = de2bi(in\_dbit(jj),k)';

if enab == 1

out\_bits = [xor(in\_bits(1),scr\_data\_rev(1)) xor(in\_bits(2),scr\_data\_rev(2)) xor(in\_bits(3),scr\_data\_rev(3)) xor(in\_bits(4),scr\_data\_rev(4))];

else

out\_bits = [in\_bits(1) in\_bits(2) in\_bits(3) in\_bits(4)];

end

scr\_dbit(jj,1) = bi2de(out\_bits);

end

end